ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
INGENIERÍA DEL SOFTWARE

REIMPLEMENTACIÓN DE LA INTERFAZ DE UN SISTEMA PARA LA EXTRACCIÓN MASIVA DE DATOS A TRAVÉS DE LA WEB

Reimplementing the interface of a Web-based Masive data extraction System

Realizado por
ALEJANDRO ALMADANA NIETO
Tutorizado por
EDUARDO GUZMÁN DE LOS RISCOS
Departamento
LENGUAJES Y CIENCIAS DE LA COMPUTACIÓN

UNIVERSIDAD DE MÁLAGA
MÁLAGA, FEBRERO 2015

Fecha defensa:
El Secretario del Tribunal

Resumen: El objetivo principal de este trabajo es rehacer la interfaz para hacerla más funcional y mejorar su usabilidad. Asimismo se mejorarán ciertos aspectos de la capa de control de la aplicación, para mejorar la interacción con la interfaz. Entre otros aspectos, en el marco de este proyecto, se realizarán las siguientes actuaciones:

Creación de un perfil administrador que permita la creación, edición, borrado y listado de todos los usuarios registrados en el sistema.

Cada usuario podrá ver las tareas que en ese momento estén en ejecución. El usuario administrador podrá hacer lo propio para todos los usuarios del sistema.

Uso de sesiones para mantener la información entre las diferentes página. Esto permitirá reorganizar la información de forma más eficiente, poder crear y borrar tareas fácilmente, etc.

Refactorización del controlador de la aplicación para mejorar la forma en la que vista y controlador se relacionan, haciendo más eficiente el flujo de información entre ambos…

En conclusión, el conjunto de requisitos tiene como objetivo mejorar la interfaz para hacer la aplicación más funcional, mejorando la experiencia del usuario, así como la implementación de nuevos requisitos que hacen que la aplicación sea más úºtil y eficiente.

Palabras clave:

Reimplementación, Manager, Crawling, Interface, Servlet, jsp, tareas, sesiones, ajax.

Abstract: This academic project is based on European project SiSOB, financed by the VII Programa Marco. Concretely, it is focused on one of the tools, which was developed on this project, the SiSOB Data Extractor, whose aim is related to extract information on Internet about researchers.

The main purpose of this project consists on re-implementing the web interface in order to make it more functional and improve the usability. Some of requirements, which will be implemented, are next ones:

Creating an admin profile, which one will allow creating, editing, removing and listing to the other registered users.

Each user will able to see the tasks in execution state and the admin user will able to see all tasks.

Using session in order to keep on the information between different web pages.

Forms validations.

Refactoring the controller in order to improve the efficiency…

All the other requirements can be consulted in detail in the main documentation of the project.

In conclusion, the main objective is improving the application in order to make better the experience for the user, adding some news functionalities.

Keywords: Reimplementación, Manager, Crawling, Interface, Servlet, jsp,
tareas, sesiones, ajax.

ÍNDICE

1 INTRODUCCIÓN

Este trabajo de fin de grado se basa en uno de los resultados del proyecto europeo SiSOB, financiado por el VII Programa Marco. Más concretamente se centra en una de las herramientas que se desarrollaron en este proyecto, el SiSOB Data Extractor, cuyo objetivo es la extracción de información en Internet sobre investigadores.

1.1 OBJETIVOS

El objetivo principal de este trabajo es rehacer la interfaz para hacerla más funcional y mejorar su usabilidad. Asimismo se mejorarán ciertos aspectos de la capa de control de la aplicación, para mejorar la interacción con la interfaz. Entre otros aspectos, en el marco de este proyecto, se realizarán las siguientes actuaciones:

- Creación de un perfil administrador que permita la creación, edición, borrado y listado de todos los usuarios registrados en el sistema.
- Cada usuario podrá ver las tareas que en ese momento estén en ejecución. El usuario administrador podrá hacer lo propio para todos los usuarios del sistema.
- Uso de sesiones para mantener la información entre las diferentes página. Esto permitirá reorganizar la información de forma más eficiente, poder crear y borrar tareas fácilmente, etc.
- Refactorización del controlador de la aplicación para mejorar la forma en la que vista y controlador se relacionan, haciendo más eficiente el flujo de información entre ambos.
- Establecimiento de los mecanismos de seguridad que eviten que cualquiera pueda hacer un acceso indebido en una página.
- Mejora en la usabilidad del mecanismo de subida de ficheros para las diferentes tareas.
- Mecanismo de trazabilidad de la aplicación a través de la API log4j.
- Control de acceso y seguridad mediante filtros java.
- Validación de formularios plugin JQuery.
- Restructuración del código, eliminación de servicios para la lógica de negocio, nueva capa de operaciones para gestionar listados de tareas, tareas, usuarios y ficheros.
- Refresco automático de listados para ver en tiempo real el estado de la tarea.

1.1.1 TECNOLOGÍAS Y HERRAMIENTAS UTILIZADAS

En la siguiente tabla se describen las principales herramientas, plataformas de programación y librerías empleadas en este proyecto. A pesar de ser una amplia lista de tecnologías y herramientas, el objetivo ha sido intentar evitar reinventar la rueda y hacer uso de los proyectos con licencia de software libre en la mayoría de los casos.

LENGUAJE PROGRAMACIÓN	
	**JAVA EE** Plataforma de desarrollo de Oracle basada en Java orientado a las arquitecturas web. Ha sido el lenguaje de programación más utilizado a lo largo de nuestra formación.
SOFTWARE	
	**NETBEANS 8.1 IDE** IDE de desarrollo de código abierto, utilizado a lo largo de nuestra formación académica, y con una extensión especializada para desarrollar aplicaciones web en Java EE. Además de plugins que ayudan a agilizar el trabajo.
	**CREATELY APP** Herramienta CASE online gratuita, utilizada para realizar los diagramas de uso, de clase o de secuencia usando la notación UML que aparecen a lo largo de esta memoria.
	**MAVEN** Se utiliza para la gestión y construcción de proyectos en Java, con Licencia Apache 2.0. Facilita el trabajo mediante el uso de un archivo XML de configuración para describir dependencias, tanto de otros módulos como de componentes externos, y cómo construir el proyecto a base de objetivos predefinidos.

BASE DE DATOS

H2 DBO

H2 es un sistema gestor de bases de datos relacionales programado en Java. Puede ser incorporado en aplicaciones Java o ejecutarse de modo cliente-servidor. Una de las características más importantes de H2 es que se puede integrar completamente en aplicaciones Java y acceder a la base de datos lanzando SQL directamente, sin tener que pasar por una conexión a través de sockets.

Está disponible como software de código libre bajo la Licencia Pública de Mozilla o la Eclipse Public License.

LIBRERÍAS JAVA

LOG4J

Herramienta muy extendida para escribir mensajes de registro. Permite múltiples configuraciones y organizar los mensajes en función de su importancia.

Java Servlets

Librería que provee los mecanismos necesarios para la implementación del controlador de nuestra aplicación web, filtrando y distribuyendo las peticiones del cliente proporcionando una respuesta según la acción que se tome.

DESARROLLO WEB

JQUERY

Es una librería de JavaScript que proporciona numerosas herramientas, desde manipular el árbol DOM[1] hasta la interacción con AJAX. Con Licencia GPL y MIT.

[1] Document Object Model – API para representar documentos HTML y XML.

BOOTSTRAP

Framework que contiene diferentes componentes webs y otros elementos de diseños combinando HTML y CSS, además de extensiones de JavaScript opcionales. Licencia Apache 2.0

1.2 CONTENIDO DE LA MEMORIA EN CAPÍTULOS

La memoria se ha estructurado de tal forma que se basa en cubrir las etapas seguidas en cualquier desarrollo de software, permitiendo al lector observar la evolución de manera iterativa del planteamiento expuesto en la introducción.

1.2.1 TECNOLOGÍAS UTILIZADAS

En esta sección de la documentación se explicará el porqué de la selección y uso que se le da a algunas de las tecnologías ya nombradas.

Entre las explicadas se encontrarán Maven, Java Servlets, JQuery y Bootstrap por ser a las que más tiempo se le han dedicado para su integración en el proyecto y por ser las bases del desarrollo.

1.2.2 ANÁLISIS DE REQUISITOS

Durante esta primera fase se describirán con más detalles los requisitos que se obtienen a partir de las diferentes reuniones mantenidas con el tutor del proyecto. Se agruparán según su función dentro del proyecto.

De cada requisito funcional obtenido en la primera fase se documentará los casos de uso más significativos. En ellos se detallará los actores, escenarios y las post/pre condiciones que deben cumplirse. A partir de ellos será posible la definición del sistema y del diseño del diagrama de clases. Serán una pieza clave para la implementación.

1.2.3 IMPLEMENTACIÓN Y PRUEBAS

En la última fase del proceso, se hará un resumen de lo más destacado en cuanto a problemas encontrados, solución propuesta y herramientas utilizadas.

1.2.4 CONCLUSIONES

En esta sección habrá un resumen de lo aprendido durante todo el proyecto, haciendo una comparación entre los objetivos y los resultados conseguidos. Se darán las conclusiones obtenidas durante el proceso de desarrollo y también de las personales.

2 TECNOLOGÍAS Y HERRAMIENTAS UTILIZADAS

En esta sección se explicará mejor la elección y el uso de las tecnologías más destacadas utilizadas que suponen un mayor beneficio para el proyecto.

2.1 MAVEN

Desarrollada por Apache Software Foundation, es una herramienta destinada a gestionar y construir proyectos de Java. Contiene una serie de ventajas a la hora de comenzar cualquier proyecto y simplemente realizando la configuración del mismo a través de un archivo XML, llamado POM (Project Object Model).

En él se especifica el software a construir, dependencias, componentes, orden de construcción entre otras. Sin contar además que tiene objetivos definidos para realizar tareas como es el caso de compilación, empaquetado, pruebas, generación de documentación. Entre las más destacadas están:

- Gestión de dependencias: a través del repositorio central se pueden importar gran cantidad de librerías y utilidades, sin necesidad de guardar cada una de ellas y despreocupando al programador de las mismas a la hora de compilar.
- Estandariza la estructura de los directorios, todos los proyectos comparten una estructura similar que ayuda a mantener el orden de los mismo, también permite configurarlos de forma personalizada.
- Integrado perfectamente con la IDE con la que se ha trabajado.
- Aporta además una implementación de ciclo de vida del proyecto, siendo las principales *compile, test, package, install, deploy*.

2.2 BOOTSTRAP

Por último, y por nombrar también algunas de las tecnologías utilizadas en el diseño de la interfaz, hay que destacar el uso de conjunto de herramientas que contiene plantillas de diseños, tipografías, botones, menús desplegables y de navegación y otra gran cantidad de elementos.

Hay que hacer especial mención en las siguientes utilidades frecuentemente utilizadas a lo largo del diseño:

- Grid System de 12 columnas que puede ser adaptable (responsive) si se requiere.
- Galería de iconos que representan multitud de acciones y son gratuitos.
- Utilización de *Modal* para desplegar diálogos y ventanas emergentes dentro de la pantalla.
- *Tooltips* para mostrar información de los botones y sus acciones.

Bootstrap nos facilita la aplicación de un conjunto de buenas prácticas, nos ayuda a adaptar una forma de trabajo, en lo que se refiere a tratar CSS + HTML5, y existen multitud de webs donde se pueden visualizar ejemplos realizados con esta tecnología y adaptar a nuestros proyectos.

3 ANÁLISIS DE REQUISITOS

3.1 ANÁLISIS

Antes de comenzar a listar los requisitos que se nos piden, los actores que tenemos en el sistema y los diversos casos de uso que vamos a tener que desarrollar, es conveniente introducir al lector en la situación que nos encontramos al comenzar el proyecto.

Tenemos una aplicación web que se encarga de crear y lanzar tareas básicamente. Un usuario se autentifica, y tiene la opción de crear una tarea, lanzarla y obtener sus resultados.

Ahora bien, después de analizar la aplicación, vemos que tiene graves defectos de seguridad y usabilidad:

-Podemos ver la contraseña y nombre de usuario de la persona que esté utilizándola tan sólo con observar la barra de navegación, podríamos copiar esa cadena de texto y desencriptarla para tener libre acceso a la misma.

-No hay ningún tipo de filtrado para acceder a recursos.

-Todos los datos son enviados por parámetros, no existen sesiones.

-Ningún elemento de la página puede actualizarse sin tener que cargar la página completa de nuevo.

-No hay mecanismo de trazabilidad.

-No existe posibilidad para crear un usuario o simplemente modificarle su contraseña de acceso.

-No podemos eliminar tareas después de crearlas.

En definitiva es una aplicación difícilmente mantenible, con una interfaz pobre, una usabilidad nula, y una falta de seguridad muy grave.

Partiendo de este análisis, hemos obtenido un listado de requisitos para transformar la aplicación e implementar soluciones que cubran todas las necesidades del usuario

3.2 REQUISITOS

3.2.1 REQUISITOS FUNCIONALES

Aquí se van a listar los requisitos funcionales, que son los que determinan la nueva funcionalidad de la aplicación y definen las acciones que pueden realizar los diferentes roles de la misma. Vamos a centrarnos únicamente en los requisitos relacionados con las nuevas funcionalidades, obviando las funcionalidades ya desarrolladas ya que no forman parte de este TFG. Las listas están divididas según el rol para poder llevar una trazabilidad.

Vamos a comenzar listando los requisitos funcionales exclusivos para un perfil de usuario administrador:

Categoría	ID	Nombre	Descripción
ADMINISTRADOR	A01	Ejecutar tarea admin	El administrador podrá ejecutar tareas de cualquier otro usuario.
	A02	Eliminar tarea admin	El administrador podrá eliminar tareas de cualquier otro usuario.
	A03	Alta usuario	El administrador podrá dar de alta usuarios en el sistema.
	A04	Editar usuario	El administrador podrá editar cualquier usuario dado de alta en el sistema, incluido a sí mismo.
	A05	Eliminar usuario	El administrador podrá eliminar cualquier usuario dado de alta en el sistema.

Para un usuario estándar se tendrán en cuenta los siguientes requisitos:

Categoría	ID	Nombre	Descripción
USUARIO ESTÁNDAR	S01	Cambiar contraseña	Los usuarios estándar podrá cambiar su contraseña de acceso la aplicación.
	S02	Eliminar tarea	Los usuarios estándar podrá eliminar tareas propias.

3.2.2 REQUISITOS NO FUNCIONALES

Estos requisitos determinan restricciones o condiciones sobre las que se ejecuta o desarrolla el sistema. La lista de los mismos se hará en la tabla con una categoría del requisito, identificador y descripción:

Categoría	ID	Descripción
Seguridad	RN01	No se permitirá acceso a ningún recurso sin autenticación previa.
Seguridad	RN02	No se permitirá que usuarios ajenos al sistema tengan acceso.
Usabilidad	RN03	El sistema informará del resultado de realizar cualquier acción a través de mensajes por pantalla.
Usabilidad	RN04	El sistema controlará los datos introducidos en los campos para evitar datos erróneos.
Interfaz	RN05	El sistema mostrará de una zona de administración para el rol administrador donde podrá ver tareas y usuarios en listados y llevar a sus funcionalidades previamente descritas fácilmente
Accesibilidad	RN06	El sistema podrá utilizarse en las versiones más recientes de los navegadores web Chrome, Firefox, Internet Explorer y Safari.

3.3 ACTORES

Describimos a continuación brevemente en qué consiste cada actor/rol del sistema.

- **Usuario estándar:** Este actor representa cualquier persona que tenga acceso a la utilización de los extractores con perfil estándar.
- **Administrador:** Es otro rol de la aplicación web; representa a las personas que pueden realizar tareas de mantenimiento o que trabajen directamente sobre los datos extraídos de la aplicación.
- **Sistema:** Representa la máquina donde estará ejecutándose la aplicación

3.4 DIAGRAMA DE CASOS DE USO

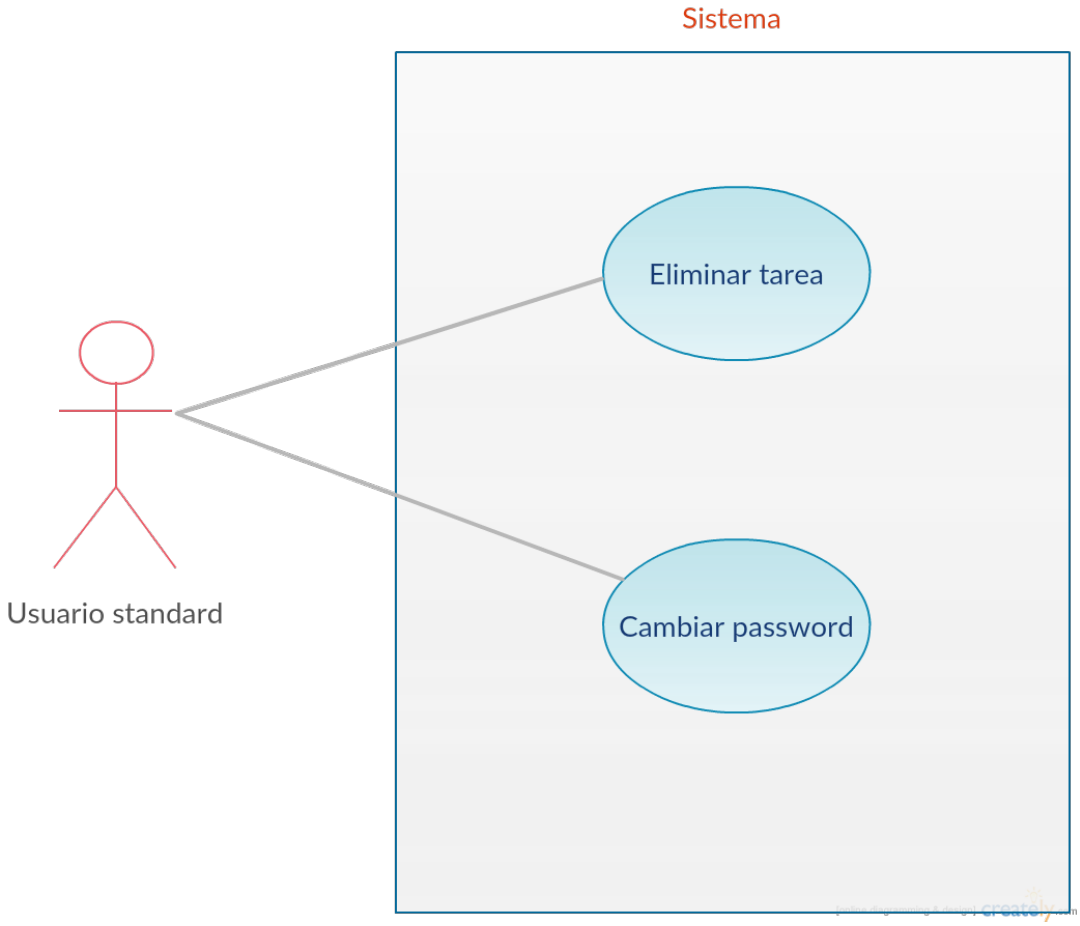


Ilustración 3.1: Diagrama de caso de uso "Usuario estándar".

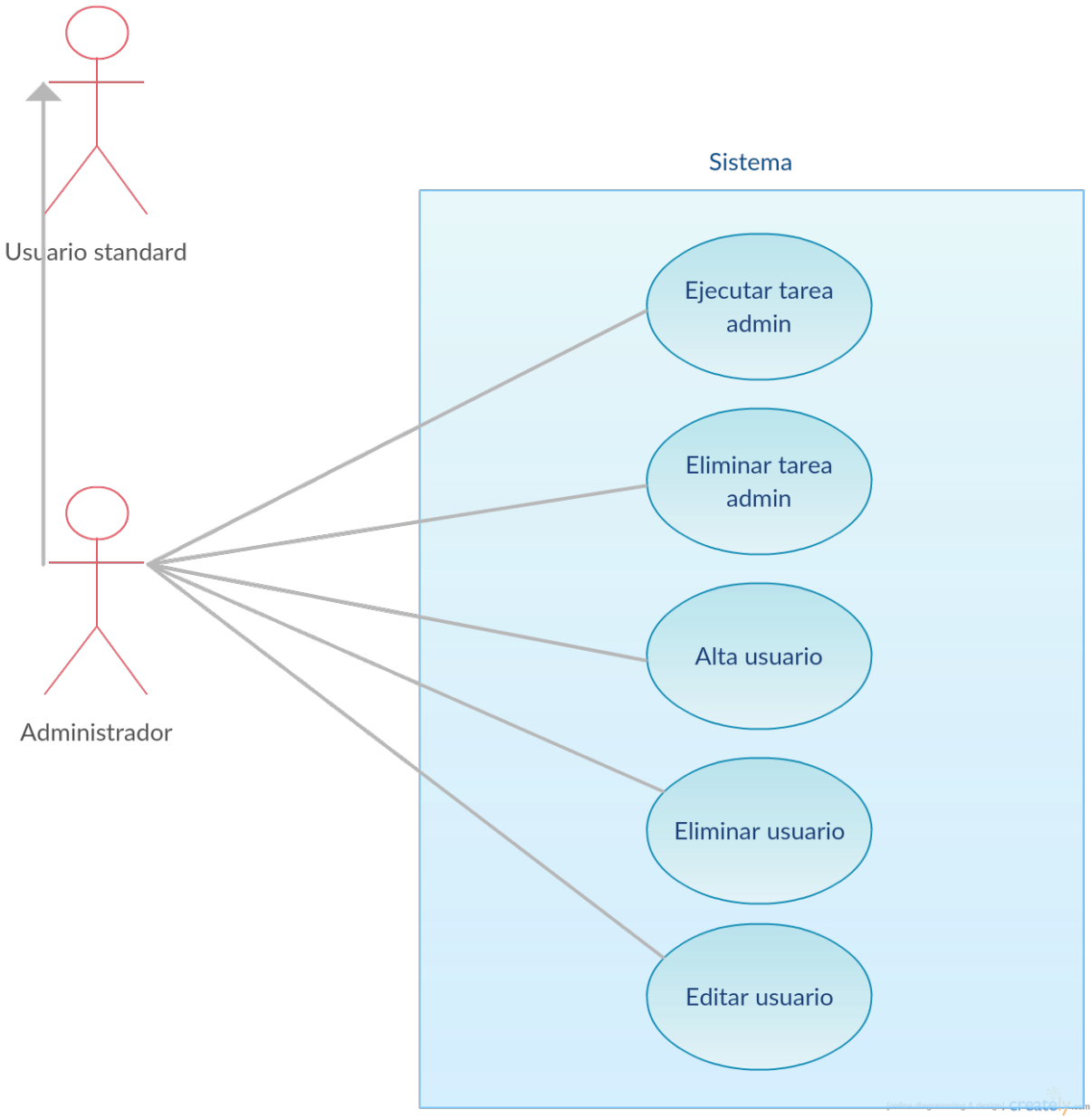


Ilustración 3.2: Diagrama de caso de uso "Usuario administrador".

4 Diseño

4.1 DESCRIPCIÓN DE LOS CASOS DE USO

En las siguientes tablas se recoge de forma detallada los casos de uso, no están expuestos todos los escenarios pero sí los más destacados.

A01	Ejecutar tarea admin	El administrador podrá ejecutar tareas de cualquier otro usuario.
A02	Eliminar tarea admin	El administrador podrá eliminar tareas de cualquier otro usuario.
A03	Alta usuario	El administrador podrá dar de alta usuarios en el sistema.
A04	Editar usuario	El administrador podrá editar cualquier usuario dado de alta en el sistema, incluido a sí mismo.
A05	Eliminar usuario	El administrador podrá eliminar cualquier usuario dado de alta en el sistema.

ID	**U1**
Caso de uso	Alta usuario
Actores	Usuario administrador
Descripción	Creación de un usuario
Precondiciones	Estar autenticado con perfil administrador
Escenario principal	1. El usuario hace clic sobre etiqueta o icono de Admin Zone. 2. El sistema muestra una ventana que contiene el listado de tareas y usuarios existentes en el sistema. 3. El usuario hace clic en botón “Add new user”. 4. El sistema muestra una ventana con un formulario para la creación de usuarios. 5. El usuario rellena correctamente el formulario y pulsa en “Add user”. 6. El sistema informa que el usuario ha sido actualizado correctamente.
Escenario alternativo	7. El sistema informa del error ocurrido al crear al usuario.
Requisito asociado	A03

ID	**U2**

Caso de uso	Editar usuario
Actores	Usuario administrador
Descripción	El usuario accede a su perfil y pulsa guardar.
Precondiciones	Estar autenticado en la aplicación con perfil administrador.
Escenario principal	1. El usuario modifica cualquier campo editable. 2. El sistema informa que el usuario ha sido actualizado correctamente.
Escenario alternativo	3. El sistema informa que ha ocurrido un error al editar usuario.
Requisito asociado	A04

ID	**U5**
Caso de uso	Eliminar usuario
Actores	Usuario administrador
Descripción	El administrador elimina a un usuario.
Precondiciones	Estar autenticado en la aplicación con rol administrador.
Escenario principal	1. El administrador elige un usuario y hace clic sobre botón “delete”. 2. La aplicación despliega un diálogo pidiendo confirmación. 3. El sistema informa a través de una notificación por pantalla de que el usuario se ha eliminado correctamente.
Escenario alternativo	2. El sistema retorna a la zona de administrador.
Requisito asociado	A05

ID	**U6**
Caso de uso	Cambiar contraseña
Actores	Usuario estándar, Usuario administrador
Descripción	Cambio de contraseña para el login.
Precondiciones	Estar autenticado en la aplicación.
Escenario principal	1. El usuario hace clic sobre su nombre de usuario ubicado en la esquina superior derecha. 2. El sistema muestra una ventana con un formulario para la edición de usuarios. 3. El usuario introduce su contraseña y la confirmación. 4. El usuario pulsa botón Update User. 5. El sistema informa de que los cambios se han realizado con éxito
Escenario alternativo	El sistema muestra un mensaje de error.

Requisito asociado	AW5

ID	**U7**
Caso de uso	Ver tarea
Actores	Usuario stándar, Usuario administrador
Descripción	Muestra la tarea perteneciente al usuario autenticado en la aplicación.
Precondiciones	Estar autenticado en la aplicación. Haber realizado U6.
Escenario principal	1. El usuario pulsa botón “View”. a. Aparece dialogo indicando que se está cargando la tarea. 2. El sistema carga al final de la página la tarea seleccionada.
Escenario alternativo	3. Se produce un error al cargar la tarea. a. El sistema devuelve un mensaje de error.
Requisito asociado	AW1

ID	**U8**
Caso de uso	Ver tarea admin
Actores	Usuario administrador
Descripción	Ver información de cualquier tarea creada en el sistema sea quien sea el propietario.
Precondiciones	Estar autenticado en la aplicación. Haber realizado U6.
Escenario principal	1. El administrador accede a la “zona admin”. 2. El sistema carga al final de la página la tarea seleccionada. 3. Se produce un error al cargar la tarea.
Escenario alternativo	- El sistema devuelve un mensaje de error.
Requisito asociado	AW10

ID	**U9**
Caso de uso	Eliminar tarea

Actores	Usuario estándar, Usuario administrador
Descripción	El usuario elimina una de sus tareas.
Precondiciones	Estar autenticado en la aplicación.
Escenario principal	1. El usuario escoge una tarea y pulsa el botón “Delete”. 2. El sistema pide confirmación. 3. El sistema muestra un mensaje de éxito y se actualiza el listado de tareas.
Escenario alternativo	4. El sistema muestra un mensaje de error.
Requisito asociado	AW16

ID	**U10**
Caso de uso	Eliminar tarea admin
Actores	Usuario administrador
Descripción	El usuario administrador accede a “Zona admin” y elimina una tarea.
Precondiciones	Estar autenticado en la aplicación.
Escenario principal	1. El administrador escoge una tarea y pulsa el botón eliminar. 2. El sistema pide confirmación. 3. El usuario acepta. 4. La tarea se elimina, se muestra mensaje de éxito y se actualiza el listado de tareas.
Escenario alternativo	5. La tarea no se elimina, se muestra mensaje de error. 6. El usuario pulsa cancelar.

4.2 MODELO DE CLASES

Tras realizar el análisis de la aplicación, se encontró que no había una entidad que representara a los usuarios ni a las tareas, sino una clase persistente que representaba el "estado" del usuario y la tarea, estas son, UserAttributes y OutputTaskStatus, representando respectivamente a usuarios y estado de tarea.

La clase que representa el estado de la tarea la seguimos utilizando por dependencia con otros módulos que no forman parte de este proyecto. Respecto a los usuarios, se ha creado una entidad usuario que es la que utilizaremos para almacenar al usuario en sesión con toda su información.

No se ha visto necesidad de definir ninguna nueva entidad para la reimplementación de la interfaz. Básicamente tendremos una clase “User”, que representará al usuario en sesión y las demás clases que conforman la lógica de negocio del sistema, que a su vez serán todas clases "controladoras". Con el siguiente diagrama y la posterior explicación, comprenderemos mejor el modelo.

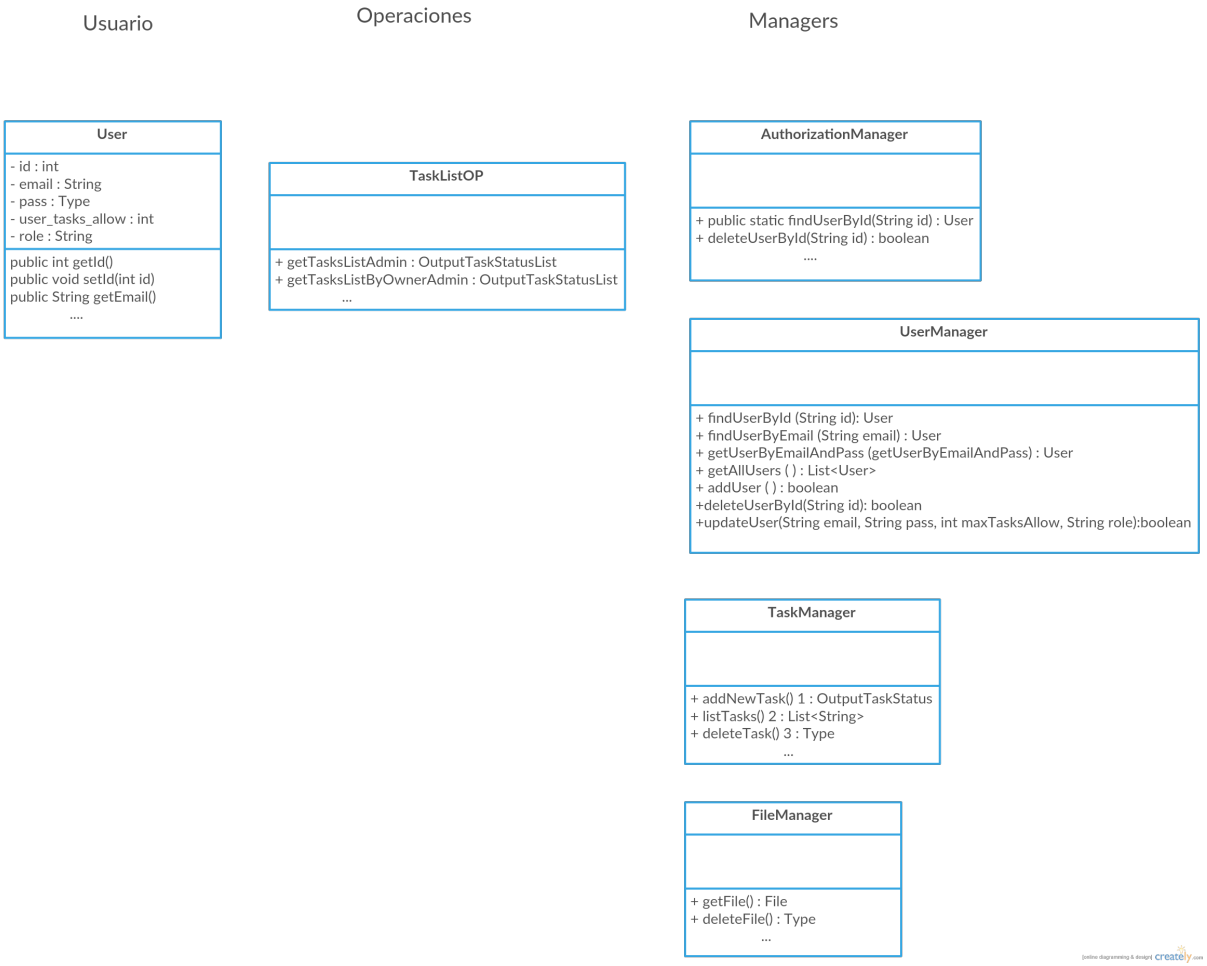


Ilustración 4.1 Diagrama de clases

-Como puede verse, tenemos la clase User, la cual utilizamos para almacenar los datos del usuario actual, y una instancia de esa clase, será el objeto que va en sesión, esta clase no tiene relación con ninguna otra, el motivo es el siguiente:

-Los manager, son todos controladores, son clases sin atributos que tienen métodos estáticos para operar cada uno de ellos respectivamente con: usuarios (crear usuarios, editar, eliminar), tareas (eliminar tareas, obtener su estado), Ficheros (son los ficheros de cada tarea, hay métodos para obtener, borrar, limpiar directorio de usuario...) y por último, AuthorizationManager, que provee de métodos para realizar validaciones contra BD, **todos ellos son la capa final que interactúa con BD y son invocados por los servlets**.

También hemos creado una clase intermedia para operar con listas de tareas (TaskListOp.java). Esta clase define métodos de tratamiento de tareas más específicos; en estos métodos se llama a TaskManager, la cual especializa el funcionamiento de TaskManager.

El funcionamiento por capas es:

CLIENTE		SERVIDOR		
JQuery&JavaScript		**SERVLETS: recogen y distribuyen peticiones**		**MANAGERS: Operan**
Llamada AJAX a un Servlet	➔	El servlet correspondiente (UserServlet,TaskServlet, FileServlet,AuthServlet) recibe la petición y llama al controlador que necesite.	➔	UserManager, TaskManager, FileManager, AuthorizationManager interactúan con BD , Usuarios o ficheros

Como podemos comprobar no hay dependencias entre clases; los manager son clases que nunca se instancian, tienen métodos estáticos que actúan a modo de clases auxiliares a las cuales llaman los servlets para hacer operaciones.

5 IMPLEMENTACIÓN

La implementación se ha llevado a cabo en 2 iteraciones; cada una de ellas ha generado como resultado un entregable.

En esta sección detallaremos los desarrollos incluidos en cada iteración así como los problemas surgidos en cada una. También incluiremos diagramas de secuencia para explicar con mayor claridad lo que realmente hace la aplicación y que facilitará la comprensión de las soluciones planteadas.

Fase	Tarea
1ª ITERACIÓN	1. Remodelación de la interfaz
	2. Reestructuración del código
	3. Traza completa de la aplicación por capas con log4java.
	4. Control de acceso y seguridad mediante filtros java.
2ª ITERACIÓN	5. Creación de un perfil administrador que permita la creación, edición, borrado y listado de todos los usuarios registrados en el sistema.
	6. Eliminación de tareas por usuarios.
	7. Mejora en la usabilidad del mecanismo de subida de ficheros para las diferentes tareas

5.1 PRIMERA ITERACIÓN

5.1.1 REFACTORIZACIÓN DE LA INTERFAZ. ESCENARIO INICIAL

El problema principal encontrado a la hora de empezar este TFG es que se basaba en un proyecto anterior con una estructura prestablecida para funcionar únicamente de la forma que lo hacía, dando muy pocas posibilidades al usuario. De hecho, la aplicación era totalmente estática y lineal, sin posibilidad alguna de gestión; lo único que nos permitía era tener un usuario, que se autenticaba en el sistema, creaba una tarea, (la cual no podía ser eliminada) y la lanzaba.

A continuación vamos a describir este escenario desde el punto de vista técnico, para posteriormente mostrar las soluciones propuestas.

Nos encontramos una aplicación que no usa sesiones para mantener la información persistente a nivel de aplicación. Información como usuario o tarea actual entre página y página, lo que se hacía era únicamente una validación inicial contra base de datos del usuario.

Al no disponer de ningún mecanismo para mantener la información de usuario y tareas entre páginas, el principal problema que acarreaba era la poca eficiencia de la aplicación. Para cada acción se hacía una nueva validación contra base de datos para comprobar de nuevo que el usuario era el correcto: para lanzar una tarea, para comprobar el estado de una tarea, para crear una nueva, para ver una tarea, etc. La aplicación básicamente funciona redireccionando a cada JSP pasando toda la información que necesite por parámetros, por eso vemos en la barra de navegación constantemente los parámetros usuario y contraseña que se utilizan para validación. En definitiva, a nivel de usuario que utiliza la aplicación web, esta web era totalmente estática, recargas de página para ver cualquier cambio reflejado, redirecciones y parámetros innecesarios y poca seguridad (se muestran usuario y contraseña como parámetros),

5.1.2 REFACTORIZACIÓN DE LA INTERFAZ. SOLUCIÓN PROPUESTA

Una vez que tenemos claros los puntos sobre los que trabajar, proponemos soluciones:

1. **Uso de sesiones:**

   Queremos modificar el funcionamiento a nivel de interfaz, que la información no se envíe constantemente por parámetros y que las páginas no se recarguen cada vez que se realice una acción, el objetivo es conseguir una aplicación web dinámica y amigable para el usuario así como aumentar la seguridad de la misma.

   En esta iteración intentamos eliminar la mayor parte de los parámetros pasados entre JSPs, y comenzamos a guardar la información de usuario y tareas en sesiones.

   Esta tarea está estrechamente relacionada con la siguiente, “Reestructuración de código”, ya que para eliminar parámetros y que la funcionalidad se mantenga hay que reestructurar la mayoría de páginas, entraremos en más detalle en la sección correspondiente a la tarea de reestructuración.

2. **Comunicación AJAX-Servlet:**

   Para conseguir gestionar las peticiones del cliente y dar respuestas del servidor de forma dinámica y sin recargas de página ni redirecciones he pensado una solución utilizando JavaScript y Servlets.

   He creado un servlet para gestionar cada tipo de acción:

   -Gestiones de usuario: UserServlet.java.

   -Gestiones de tareas: TaskServlet.java.

   Adicionalmente,

   -Para gestionar peticiones de validación de usuario y autenticación: LoginServlet.java.

   -Filtros de seguridad: AuthFilter.java

   -Configuración inicial: InitialServletController.java

Básicamente, la comunicación consiste en peticiones Ajax al Servlet correspondiente, éste retorna una respuesta, la cual es insertada en forma de código html en el lugar de la página que se necesite.

Hemos añadido esta nueva capa controladora de Servlets que recibe y gestiona las peticiones del cliente.

Nuestra aplicación quedaría estructurada de la siguiente forma:

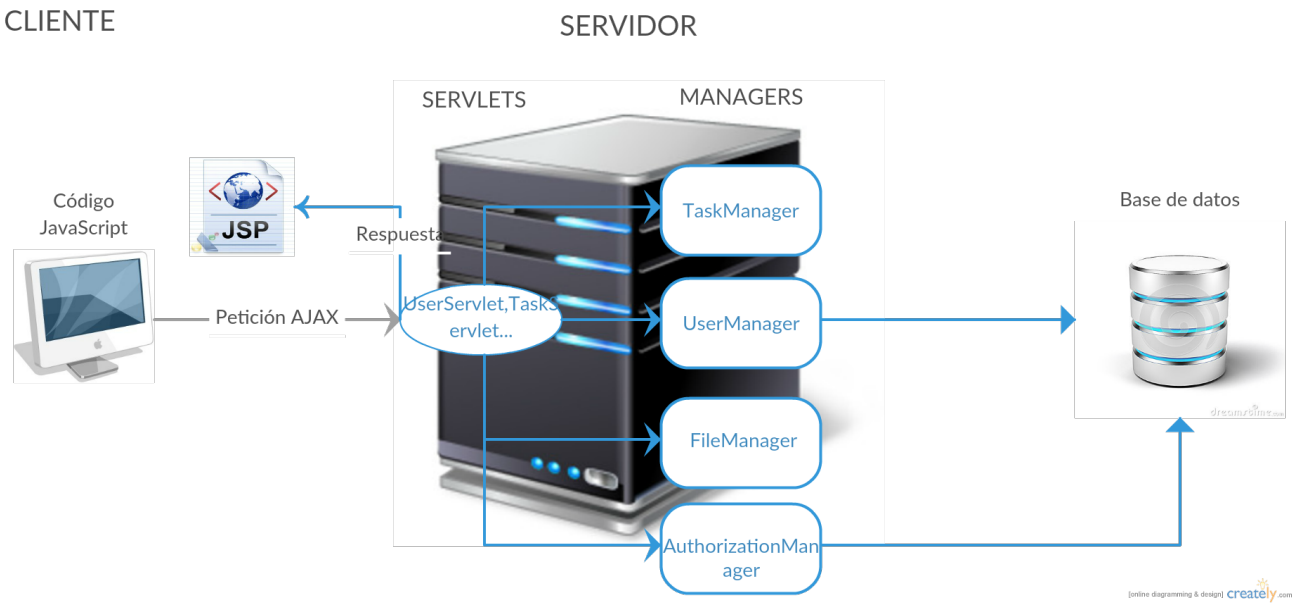


Ilustración 5.1: Diagrama despliegue.

Gracias a la tecnología Ajax, realizamos peticiones http a los Servlets y estos nos dan la respuesta la cual podemos mostrar al usuario sin necesidad de recargar la página, esto mejora sustancialmente la usabilidad de la web de cara al usuario.

En nuestro caso, con esta tecnología, podemos ver, cómo al crear una nueva tarea los listados se actualizan sin ningún tipo de espera, interrupción o de redirección a otra url.

También nos beneficiamos de esta tecnología para ver, en tiempo real, el estado de ejecución de una tarea, ya que los listados se actualizan automáticamente.

5.1.3 REESTRUCTURACIÓN DEL CÓDIGO

Tomando como referencia la ilustración, detallamos cómo hemos refactorizado y reestructurado el código de la aplicación:

Capa JSP: es donde mayor trabajo de reestructuración ha habido, hemos eliminado páginas y agrupado los JSPs por funciones, eliminado parámetros y establecido el uso de sesiones.

Capa Servlet: la hemos creado desde cero, a través de esta capa los diferentes servlet se encargan de recibir las peticiones Ajax y llevar a cabo la lógica de negocio de la aplicación.

Eliminación de servicios: las funciones que eran realizadas por servicios web han sido trasladadas a la capa controladora de los **"Managers".** Por ejemplo, todas las gestiones con tareas (creación, obtención del estado, preparación, listados…) han sido trasladadas al TaskManager.

Hemos creado una página principal: main_zone, en la cual se muestra el listado principal de tareas del usuario y hemos refactorizado los listados a una JSP, que contiene únicamente la lista de tareas, de esta manera podemos recargar fácilmente los listados mediante peticiones ajax.

5.1.4 TRAZA COMPLETA DE LA APLICACIÓN POR CAPAS CON LOG4JAVA.

El objetivo es tener un registro de las acciones que se realizan en la aplicación y llevar una trazabilidad por días, así como tener un mecanismo que nos permita identificar fácilmente cualquier error producido facilitando el mantenimiento de la aplicación, el otro objetivo es facilitarle a un futuro desarrollador su trabajo para ampliar funcionalidades de la aplicación.

La función de logging se lleva a cabo por capas, hay una ruta definida en la que se van almacenando los ficheros de log:

`/Users/apache-tomee-plus-1.7.1/bin/share/app`

Las diferentes capas son:

System.log: Lleva el log de la clase que se encarga de la inicialización del sistema, concretamente SystemManager.java, estas son las necesarias para arrancar la aplicación y que las conexiones con base de datos funcionen adecuadamente, se cree una instancia de conexión y las tareas de extracción se inicialicen correctamente.

Operation.log: contiene el log de clase encargada de realizar operaciones con listados de tareas, esta clase es TasksListOp.java.

Servlet.log: contiene el log del paquete Servlet, todas las peticiones del cliente y respuestas podrán ser monitorizadas a través de este fichero.

Application.log: si se quiere tener una traza completa del ciclo ciclo de vida de la aplicación podemos consultar este fichero, ya que en él quedan reflejados todos los logs.

5.1.5 CONTROL DE ACCESO Y SEGURIDAD MEDIANTE FILTROS JAVA.

Dentro de la especificación de Java Servlets, existe un componente llamado Java Filter.

Los filtros nos ayudan a analizar o transformar las peticiones realizadas sobre las páginas Web, ya sean JSPs o servlets. Estos filtros trabajan de forma encadenada, por lo que se puede crear una serie de filtros que van pasando el control de uno al otro.

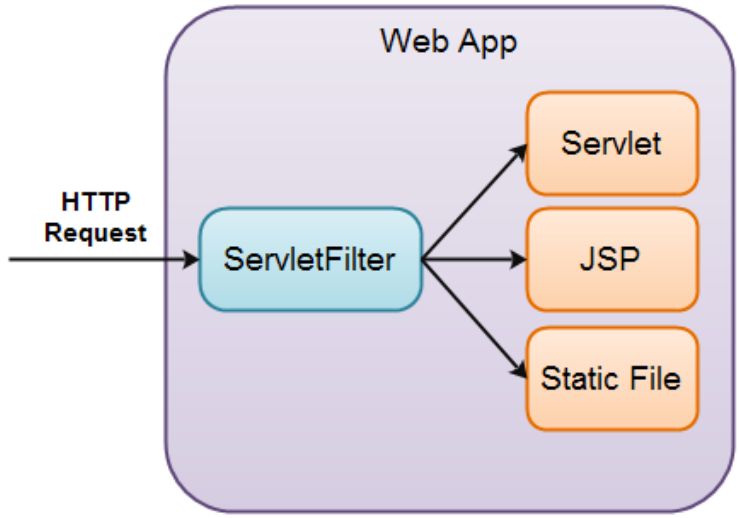


Ilustración 5.2: javax.servlet.Filter

Para generar un Filtro es necesario que la clase en cuestión herede ("inherit") el comportamiento de la clase llamada Filter, a través de dicha clase se tiene acceso a los métodos y mecanismos ofrecidos por un filtro. Este procedimiento es muy similar al de un Servlet que hereda ("inherit") su comportamiento de la ClaseHttpServlet (o GenericServlet).

Los tres métodos que debe implementar un filtro son:

doFilter (Obligatorio): Este método es la parte medular de todo Filtro ya que dentro de él se incluyen las tareas principales de ejecución.

init (Opcional) : Es un método ejecutado antes del método doFilter, su labor principal es adquirir/inicializar algún recurso que será empleado por doFilter.

destroy (Opcional): Ejecutado una vez que ha terminado el método doFilter, su labor es liberar los recursos utilizados/adquiridos en el proceso de ejecución los cuales generalmente son aquellos reservados por init.

En nuestro caso hemos modelado un filtro sencillo, el cual simplemente obliga a que el usuario esté en sesión para permitirle acceder a cualquier recurso, si no se cumple esa restricción, el filtro lo redirige a la pantalla de login.

Como la única forma de que el usuario esté en sesión es hacer login correctamente, este filtro nos viene perfectamente para restringir el acceso.

5.2 SEGUNDA ITERACIÓN

5.2.1 CREACIÓN PERFIL ADMINISTRADOR

Para este requisito hemos definido una entidad: User.

Este clase tiene los siguientes atributos:

```
private int id;
private String email;
private String pass;
private int user_tasks_allow;
private String role;
```

El más relevante es el atributo “role”, en el que almacenaremos el rol del usuario. Hemos definido dos roles: “admin” y “estándar”.

Hemos creado una zona de gestión específica para el usuario administrador:

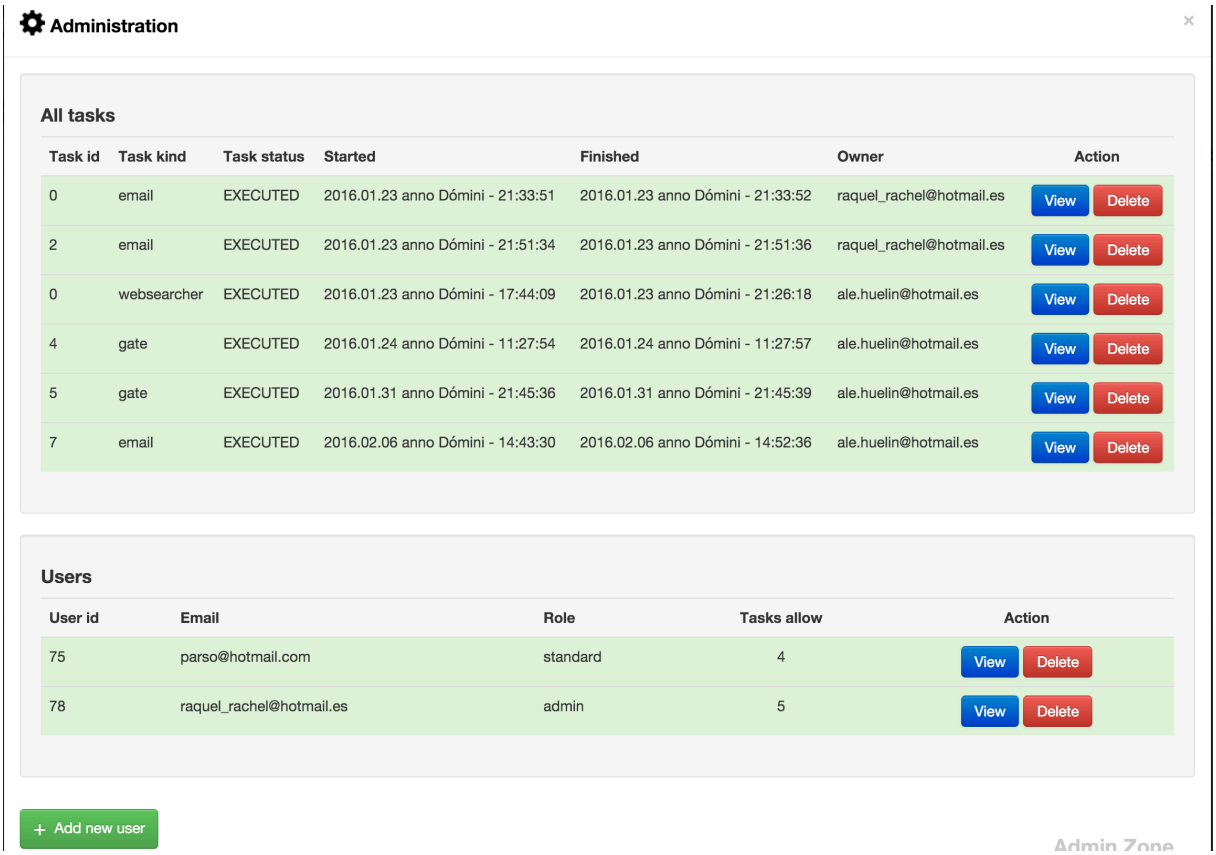


Ilustración 5.3: Admin Zone.

El usuario administrador, como podemos ver en la ilustración, tiene permiso para ver (por lo tanto, también ejecutar) y eliminar cualquier tarea, así como crear y editar usuarios.

Esta zona de administración es un nuevo fichero .jsp llamado admin_panel.jsp.

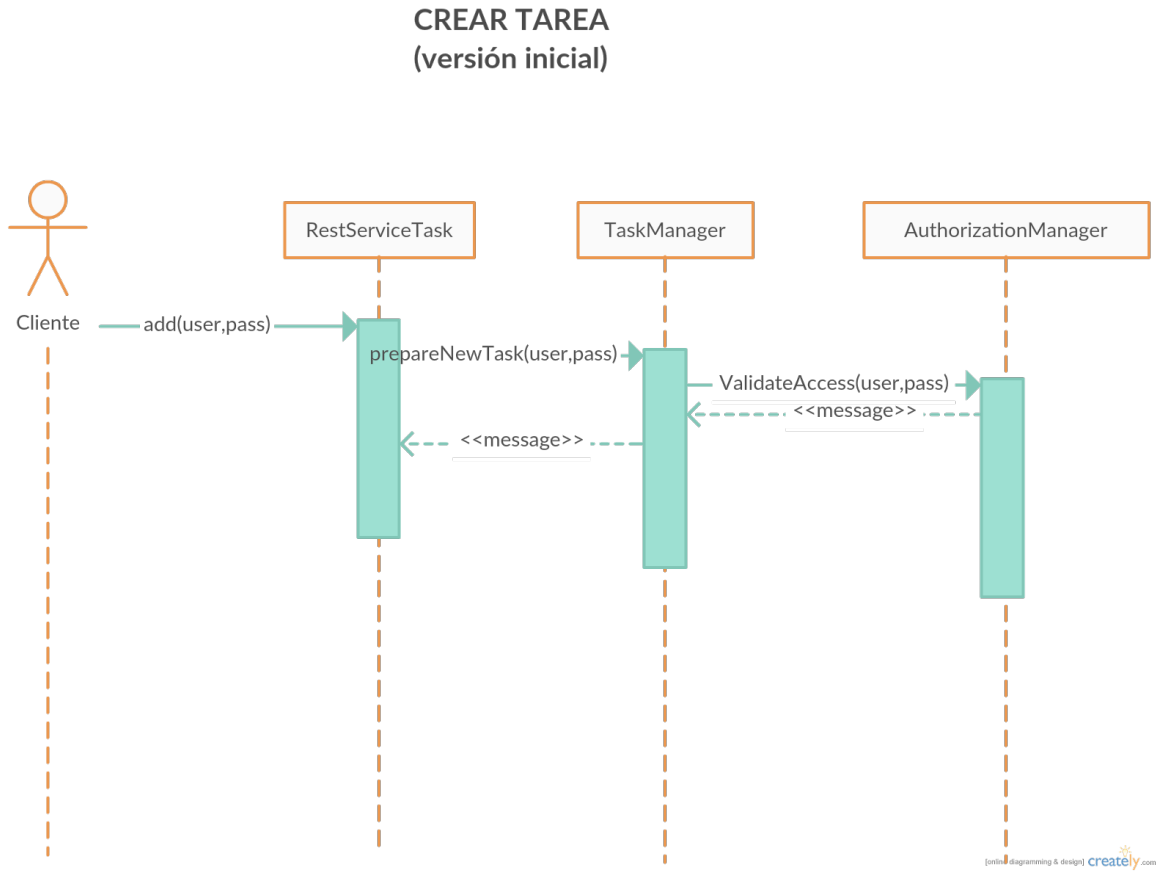
La comunicación entre capas y la forma de devolver las respuestas del servidor es común para todos los casos de uso, la variante es que en cada uno de ellos se llaman a servlet y clases managers diferentes, según estemos tratando con tareas, usuarios o fichero

Para todos los casos:

El usuario pulsa un botón, que se traduce en una petición AJAX al servidor, la cual lleva unos parámetros según qué acción se quiera realizar, esta petición es recogida por un servlet que, según el tipo de acción, invocará a una clase de la capa manager, la cual interactúa directamente con la base de datos para insertar, eliminar, o buscar información.

Para entender mejor el funcionamiento vamos a mostrarlo a través de los correspondientes diagramas de secuencia.

En primer lugar vamos a mostrar un diagrama para explicar el funcionamiento del sistema antes de la refactorización del código y de la introducción de la capa de servlets para la gestión de tareas:



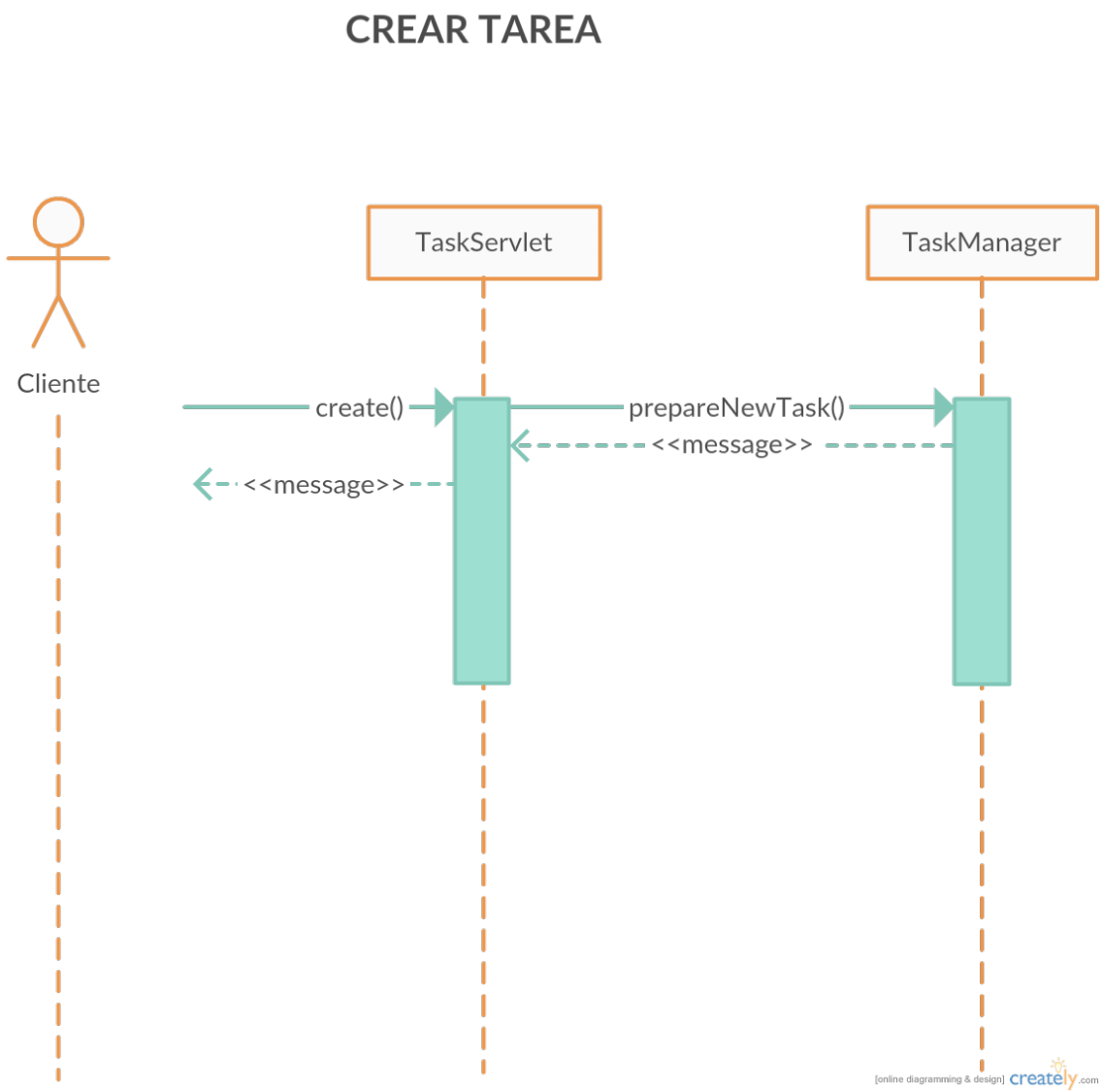
El cliente pulsa el botón crear, a partir de ahí se producen los siguientes eventos:

1. Se realiza una petición AJAX, enviando como parámetros, usuario actual y contraseña, desde el jsp a un servicio rest, espeficiando la uri add.
2. El servicio rest ejecuta la siguiente lógica de negocio:
    a. Llama al método prepareNewTask de la clase taskManager; éste método realiza una validación contra base de datos, para comprobar usuario y contraseña, si la validación es correcta, la clase TaskManager crea la tarea.

A simple vista podemos observar dos características que hacen este proceso más pesado e ineficiente, en primer lugar se está realizando una llamada innecesaria a un servicio rest, la aplicación no está pensada para que se realicen llamadas remotas desde clientes externos para la creación de tareas de un usuario determinado, por lo tanto no necesitamos publicar un servicio para la creación de tareas, por otra parte, como consecuencia de no hacer uso

de sesiones, vemos que se realiza una validación del usuario y contraseña contra base de datos para poder crear la tarea.

Con los cambios realizados, hemos suprimido la capa rest, y la validación de usuario. La capa rest la hemos cambiado por un servlet que gestiona las peticiones y la validación del usuario la realizamos una sola vez al hacer el login, desde ese momento tenemos al usuario en sesión y evitamos validaciones innecesarias. Lo mostramos en el siguiente diagrama:



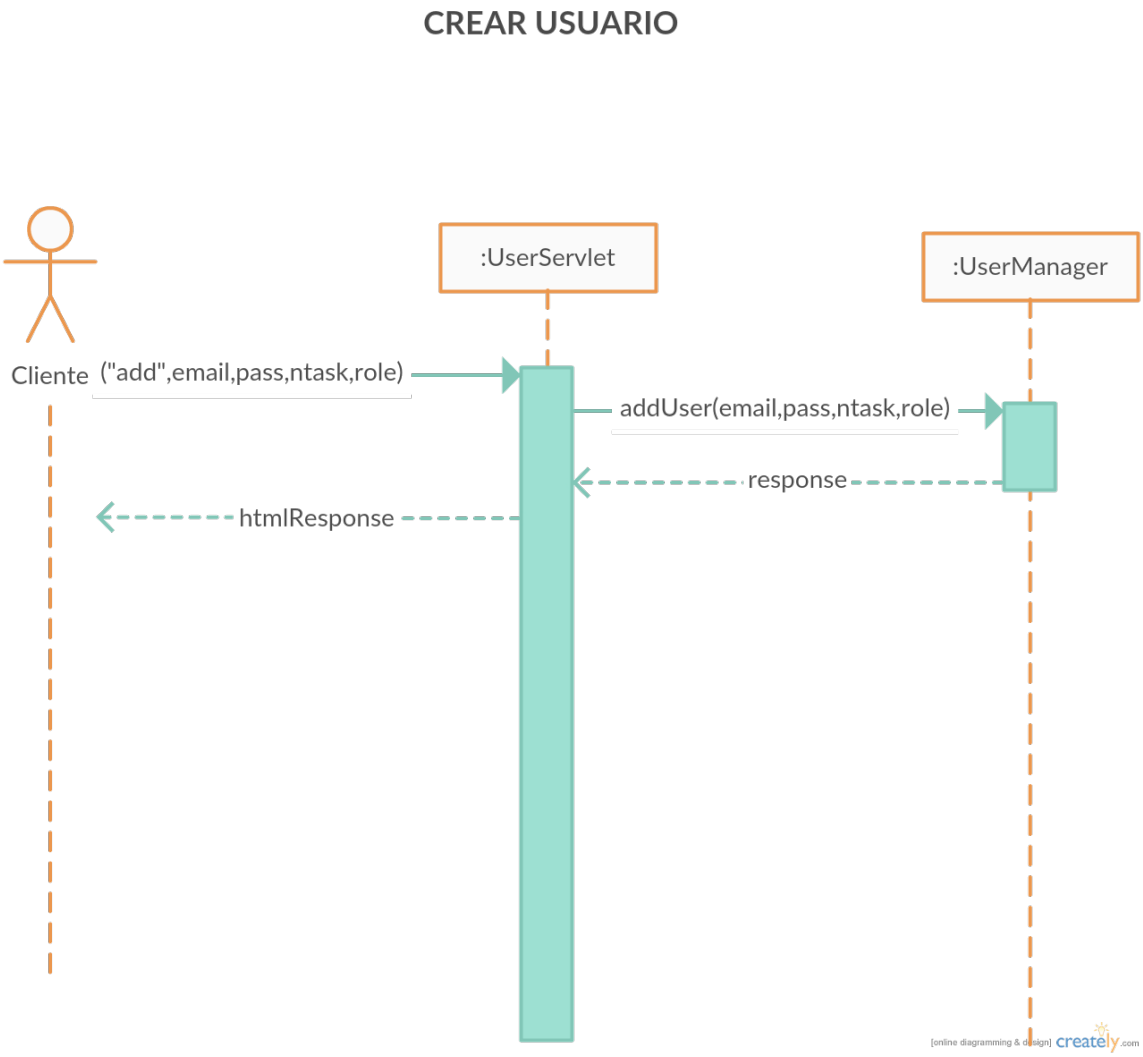


Ilustración 5.4: Add user.

El usuario introduce en el formulario los campos necesarios para dar de alta un usuario, estos son:

- Email
- Contraseña
- Numero de tareas permitidas
- Rol

Tras introducir los datos necesarios y pulsar el botón “Add user”, lo que ocurre a bajo nivel es lo siguiente:

1. Como podemos ver en el diagrama, se produce una llamada desde el “Cliente”, en este caso es un método JQuery escrito en la página JSP, al servidor.

2. Se realiza una petición AJAX, en la cual se envían: un parámetro identificador de la acción a realizar, en este caso “add”, y todos los parámetros introducidos manualmente por el usuario (email, contraseña, número de tareas y rol).

3. Esta llamada o petición AJAX del cliente es capturada por el servlet “UserServlet”, el cual recupera el parámetro identificador de la acción a realizar, “add”, y a continuación invoca a una clase encargada de la gestión de usuarios, la cual ejecuta el método que finalmente inserta al usuario en base de datos.

4. Este método que inserta al usuario en base de dato, devuelve un resultado según se haya ejecutado correctamente o no, la respuesta (“response” en el diagrama), es devuelta en cascada, primero al servlet, y luego hasta el cliente.

5. El método JQuery que ejecutó la llamada inicial es el mismo que se encarga ahora de recoger la respuesta en forma de código html (htmlResponse), que finalmente, es mostrada por pantalla, traducida a un mensaje de error o de éxito.

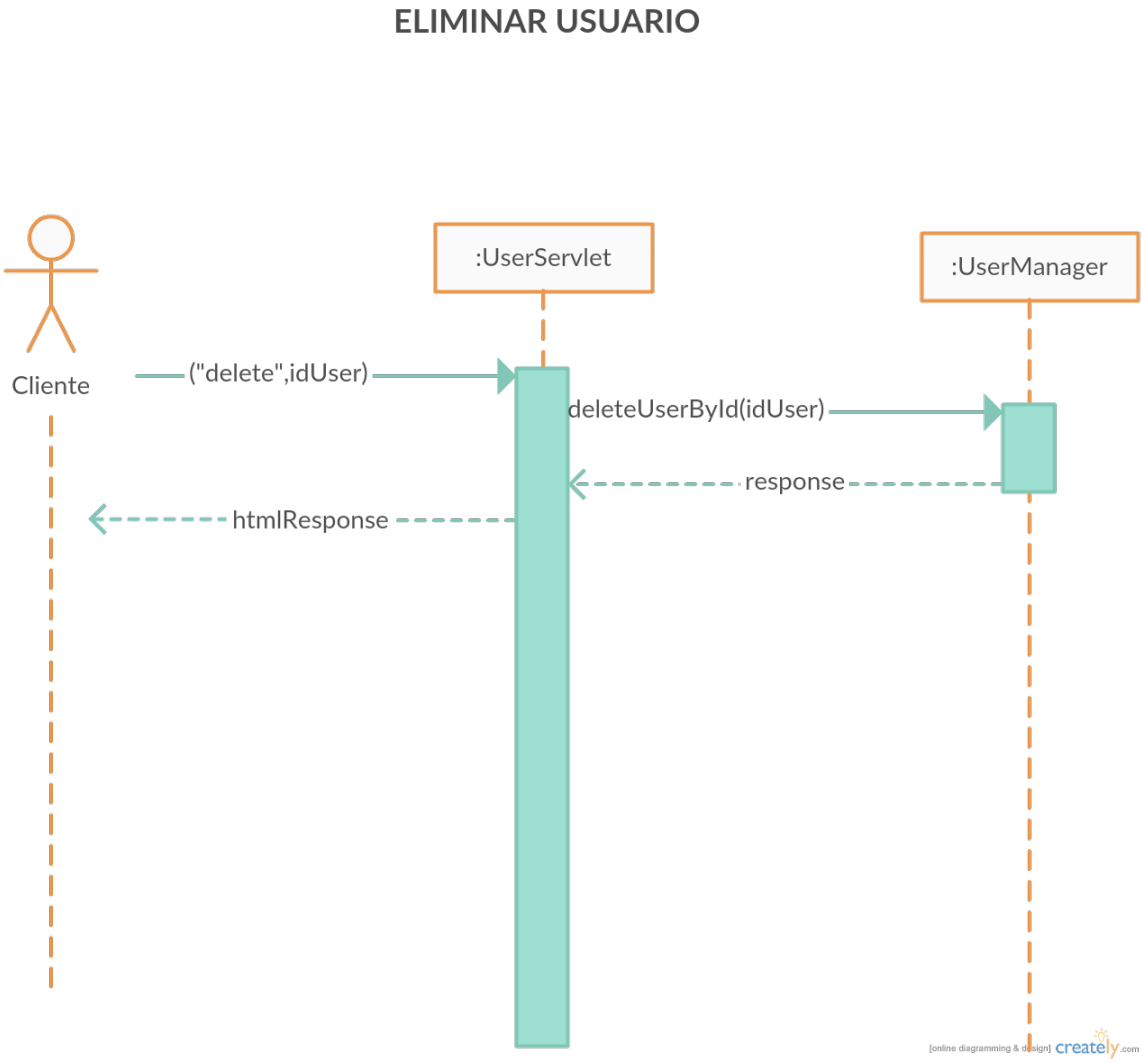


Ilustración 5.5: Delete user.

En este caso la secuencia de acciones que realiza el sistema son las siguientes:

Una vez que el usuario ha escogido al usuario que desea eliminar y ha pulsado el botón “Delete”.

1. Se ejecuta un método JQuery que se traduce en una petición AJAX, esta petición envía los parámetros “Delete” y el identificador del usuario a eliminar hacia el servlet “UserServlet”.
2. Este servlet identifica la acción a realizar, y llama a la clase UserManager, la cual ejecuta el método encargado de eliminar al usuario “deleteUserById”.
3. El método devuelve un resultado, de éxito o de error y comunica la respuesta al Servlet.

4. El Servlet devuelve la respuesta al cliente (método JQuery), la cual se traduce en una respuesta AJAX que es mostrada por pantalla en forma de código html (htmlResponse).

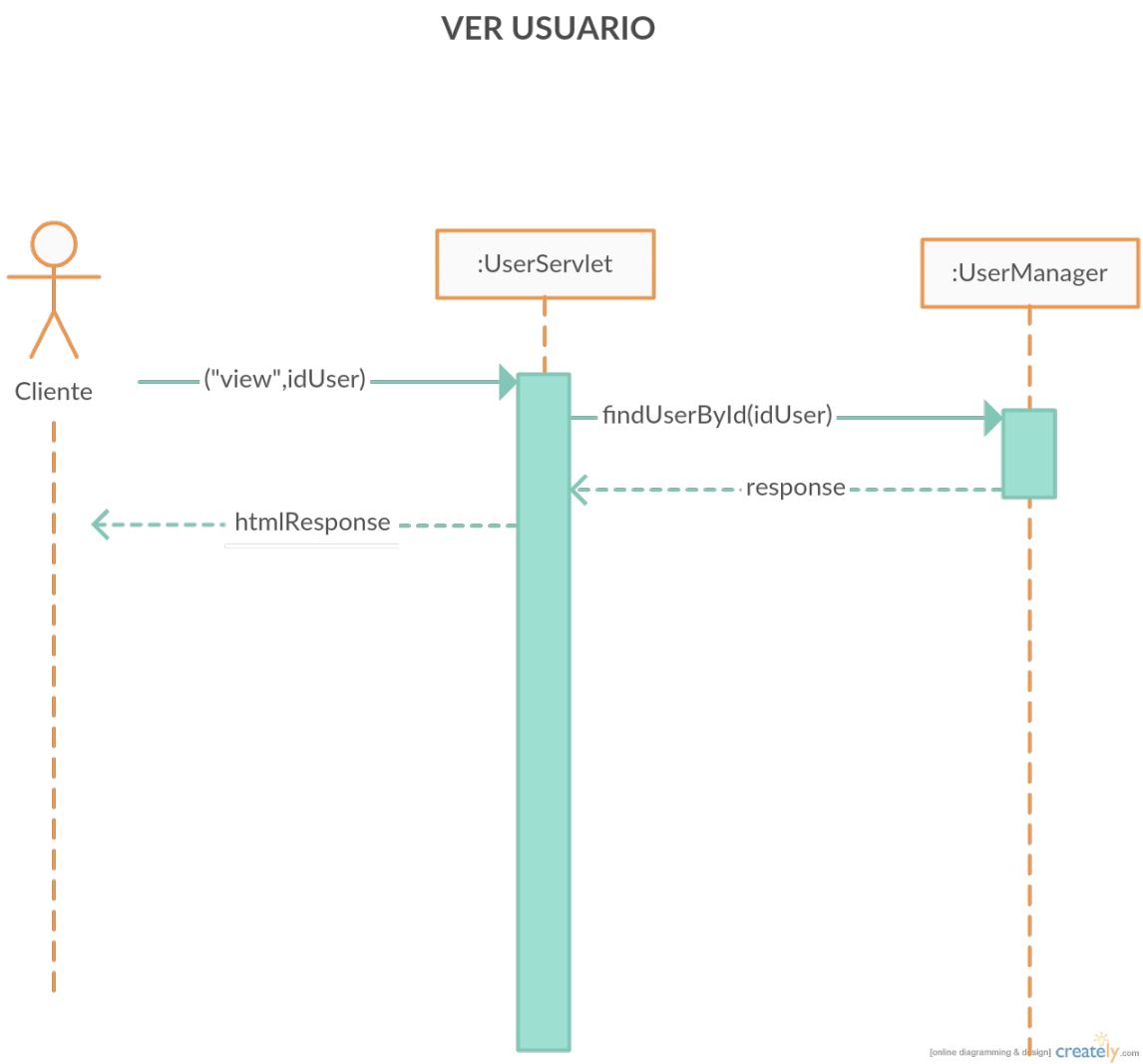


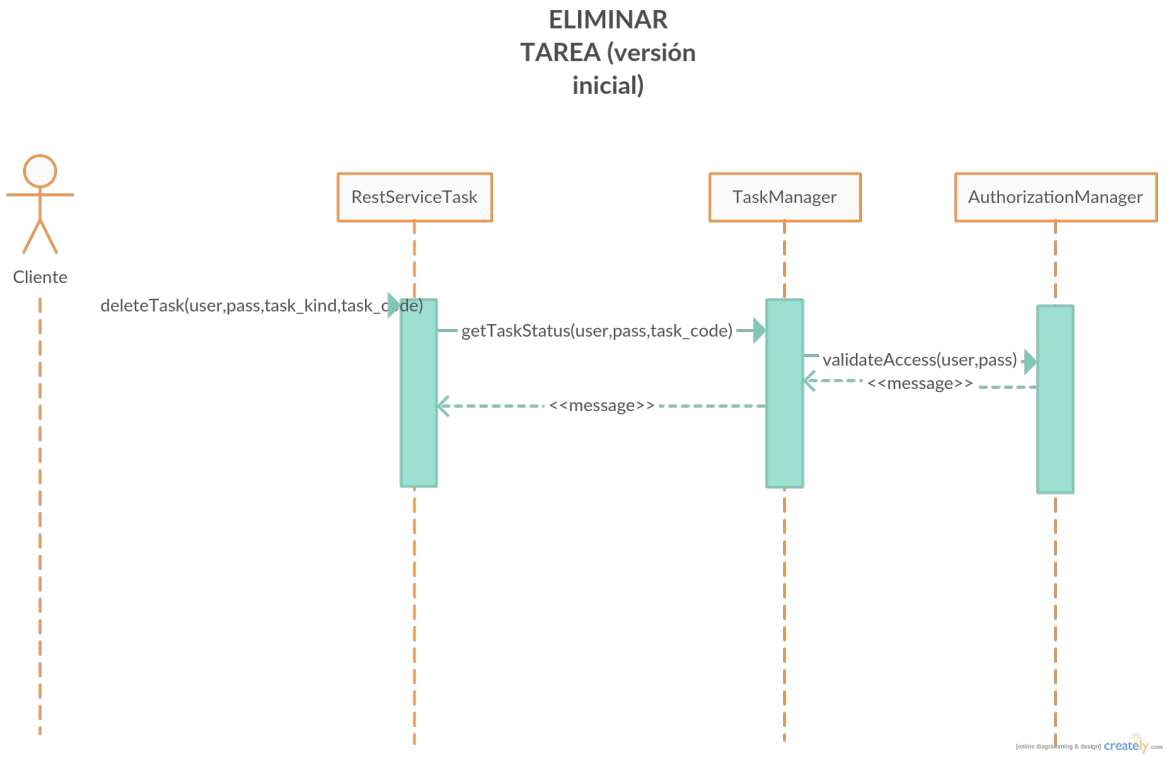
Ilustración 5.6: View user.

Funciona de forma análoga a los casos anteriores.

5.2.2 ELIMINACIÓN DE TAREAS

En su versión inicial, la eliminación de tareas no estaba completamente implementada, cuando un usuario creaba una nueva tarea, no existía un botón para eliminarla hasta que la tarea fuera ejecutada: es decir, si se creaba una tarea por error, para eliminarla había que realizar todo el proceso de lanzamiento de tareas y esperar a que terminara para eliminarla.

El proceso de eliminación también era bastante ineficiente y costoso, vamos a representarlo mediante un diagrama:



Del mismo modo que estaba implementado el requisito de crear tareas, para este requisito también se realiza una llamada al servicio rest la cual podría evitarse, y una validación del usuario y contraseña innecesaria si se tuvieran sesiones.

A continuación explicaremos cómo lo hemos hecho:

Este requisito ha sido implementado para los dos roles existentes. Los usuarios estándar pueden eliminar únicamente sus propias tareas, mientras que los administradores pueden eliminar tanto las suyas como las de los demás usuarios.

El mecanismo de interacción entre los diferentes componentes es similar al de casos anteriores:

Se genera una llamada desde el cliente Jquery al servlet controlador, en la que se envía el identificador de la tarea a eliminar y un parámetro que determina la acción a realizar, en este caso sería “delete”, a continuación, el servlet recibe este parámetro que indica la acción y también recibe un identificador, por lo tanto, ya “sabe” qué método debe llamar, entonces ejecuta el método de eliminación de clase manager correspondiente a la gestión de usuarios y se realiza el borrado en base de datos, éste método retorna una repuesta al servlet que a su vez es transmitida al cliente en forma de un mensaje de éxito o error en código html.

Vamos a representar la interacción entre clases mediante el siguiente diagrama de secuencia:

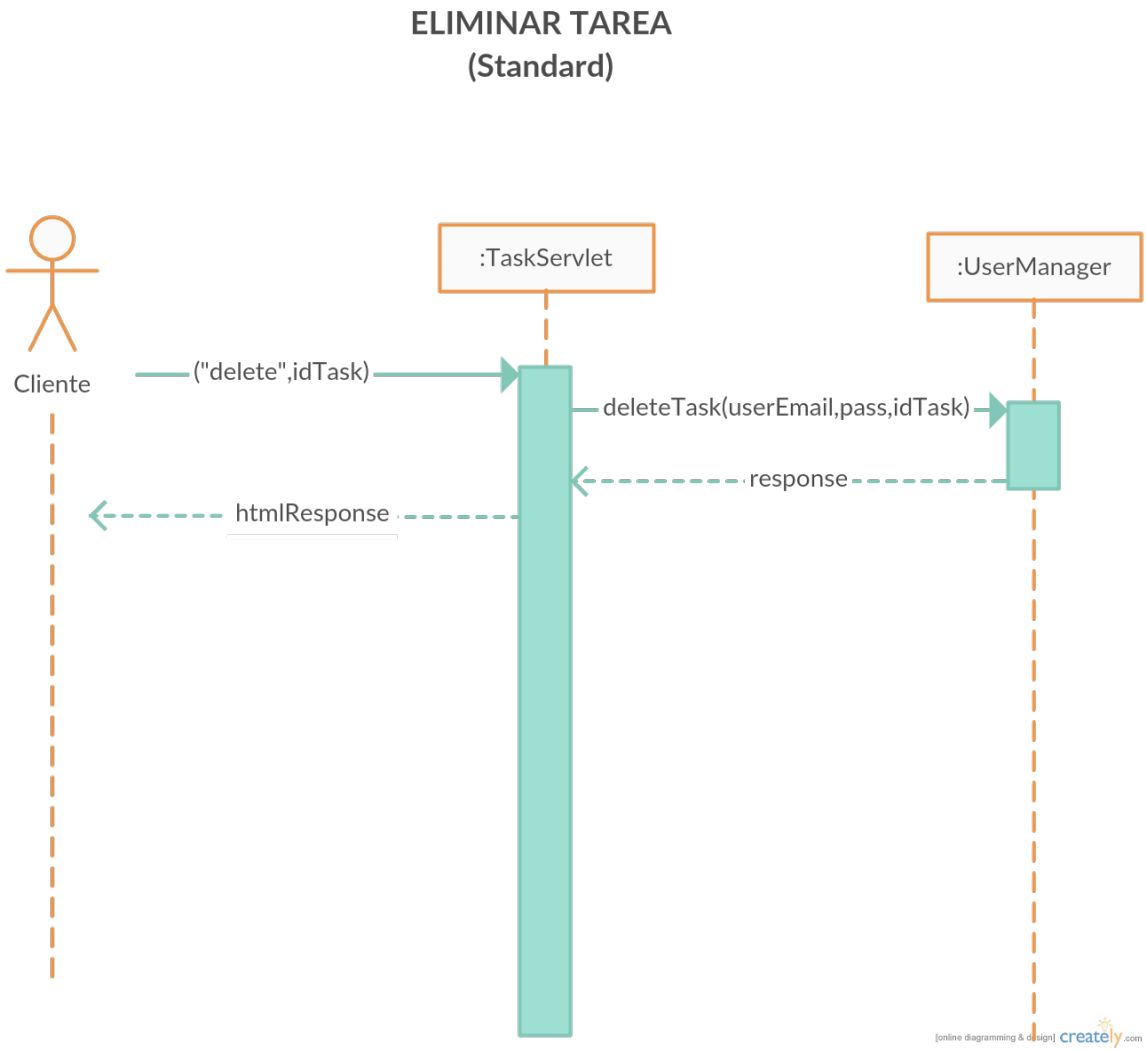


Ilustración 5.7: Delete task estándar.

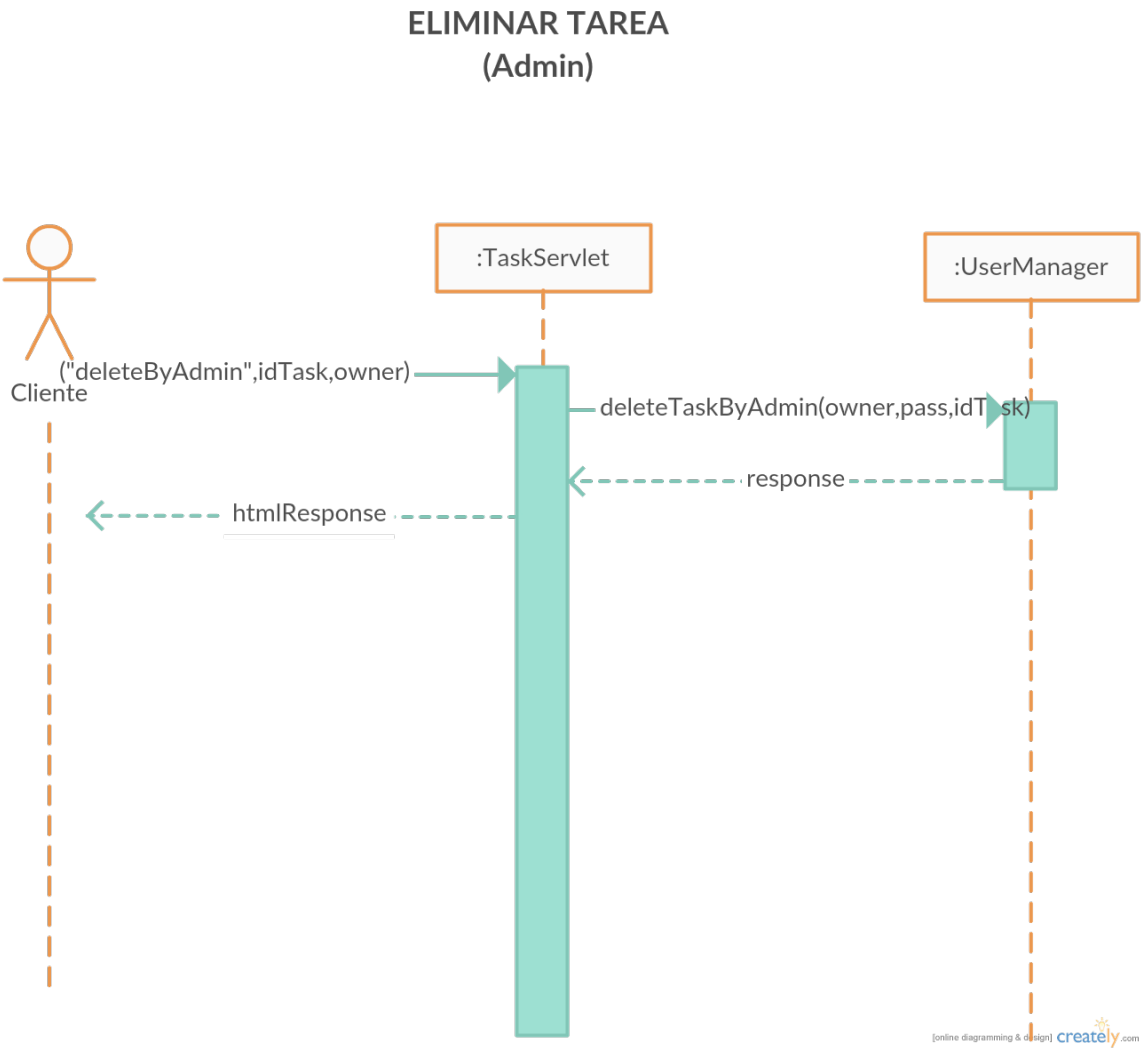


Ilustración 5.8: Delete task admin.

5.2.3 MEJORA EN LA USABILIDAD DEL MECANISMO DE SUBIDA DE FICHEROS.

Para implementar esta mejora nos hemos decantado por un plugin de bootstrap para la subida de ficheros el cual es muy sencillo se usar y atractivo estéticamente:

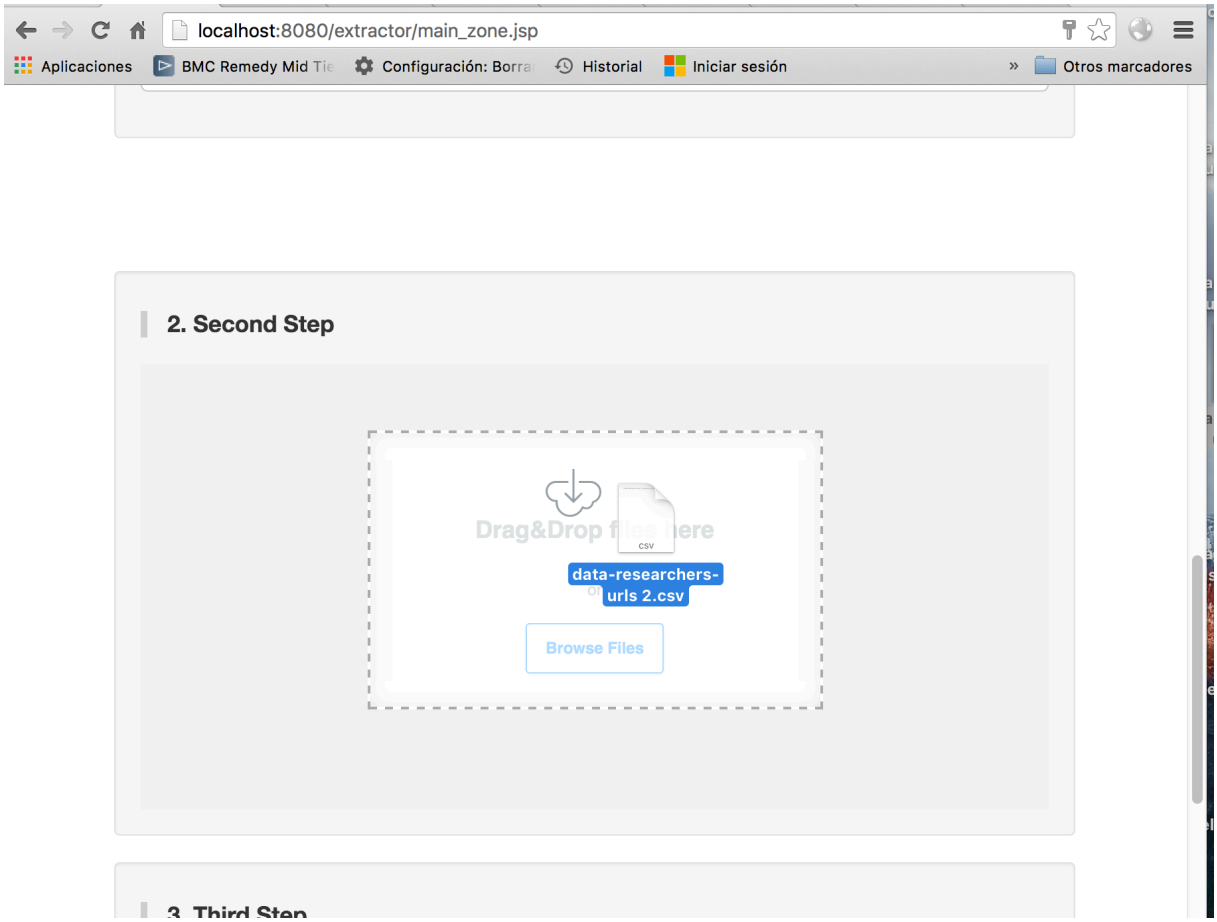


Ilustración 5.8: Uploading file.

El funcionamiento consiste en una petición jquery al servlet “UploadServlet”, el cual lee los bytes de entrada y los copia a la ruta del usuario en sesión.

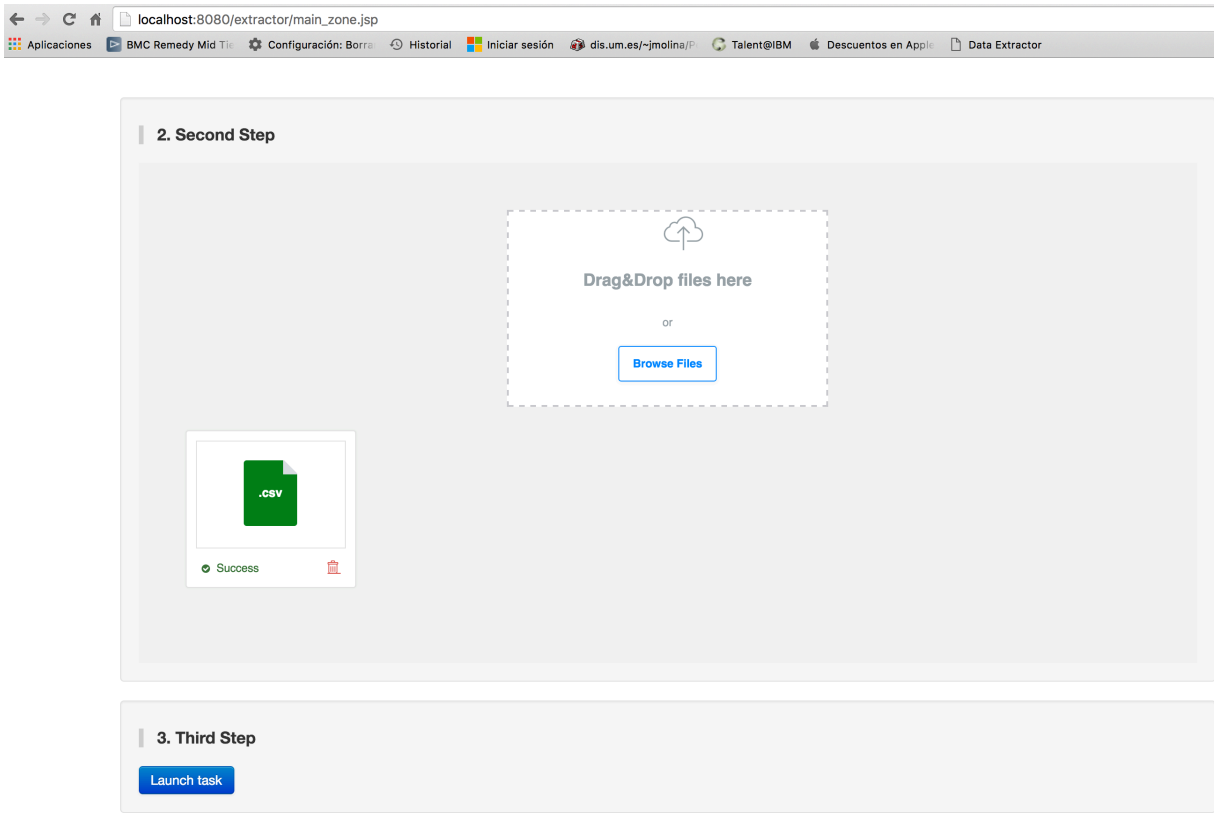


Ilustración 5.8: Uploaded file.

Una vez subido el fichero, hemos asociado el icono de eliminación del mismo a una llamada jquery al servlet de ficheros (FileServlet), esta llamada indica al Servlet que elimine el fichero del usuario, y éste llama al manager correspondiente para eliminarlo, lo vemos a continuación con un diagrama de secuencia.

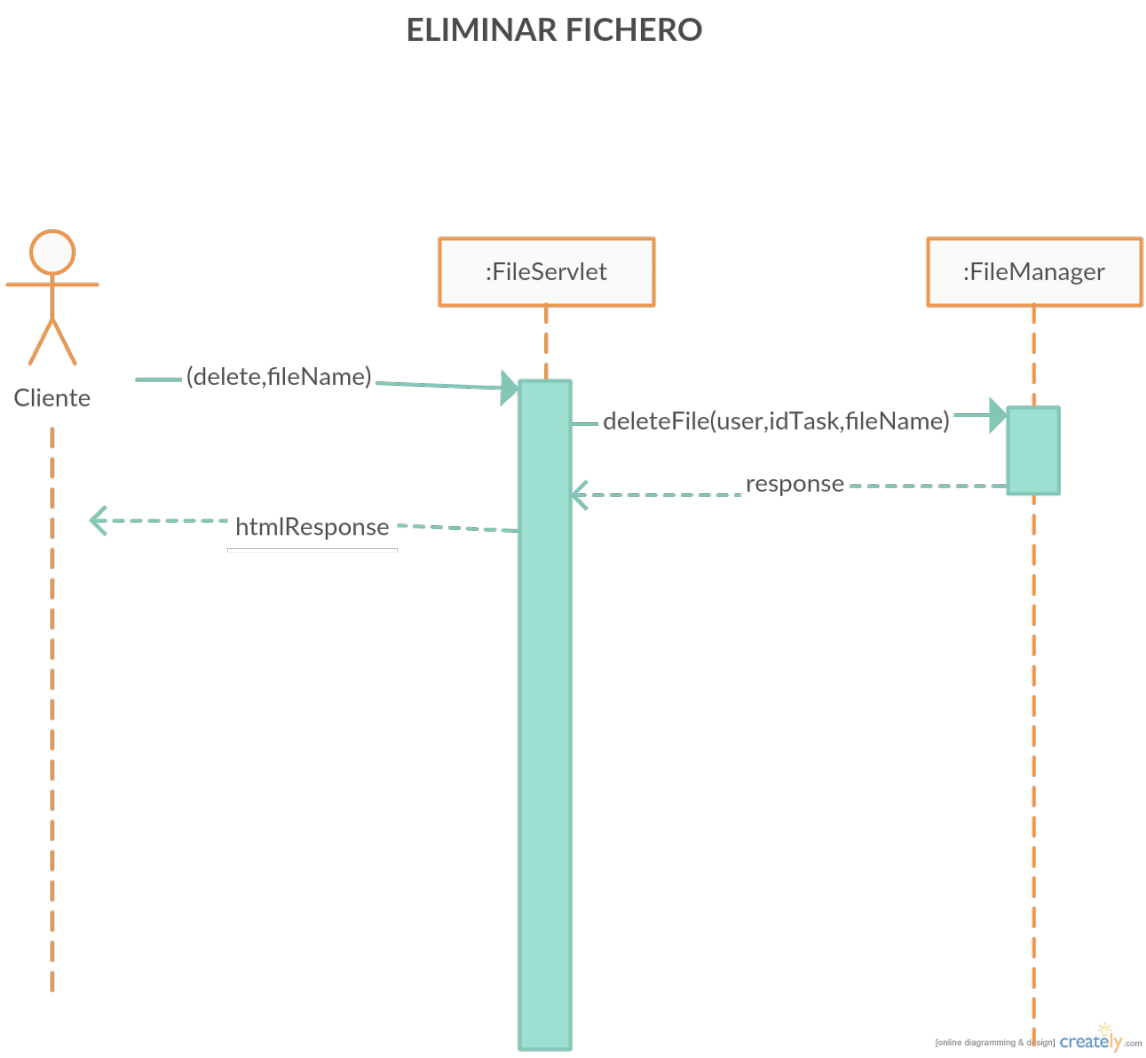


Ilustración 5.8: Deleting file.

6 CONCLUSIONES

Esta última sección está dedicada a las conclusiones obtenidas tanto de forma personal, del desarrollo y académicas en el resultado final. También se comentará futuras modificaciones pensadas para continuar el desarrollo de la aplicación.

6.1 RESULTADO FINAL Y CONCLUSIONES

Para este TFG se han propuesto una serie de tareas, para cumplir 3 objetivos principalmente:

1. Transformarla en una aplicación web que permita realizar la gestión de usuarios a través de una interfaz web:

   Nos encontrábamos con una aplicación que, básicamente, no permitía la gestión de la información con la que trata, es decir, necesita usuarios para funcionar pero no permite la creación ni la edición de los mismos por otra parte se basa en la creación y lanzamiento de tareas pero no permite la eliminación de las mismas.

2. Aumentar el nivel de seguridad:

   En la primera versión todos la información se pasaba por parámetro, podíamos ver, incluso, usuario y contraseña del usuario mirando la barra de navegación. Tampoco había un mecanismo de trazabilidad ni restricción de acceso a recursos.

3. Hacer una aplicación más usable y amigable para el usuario:

   Para cada acción sencilla el usuario debía esperar a recargar la página.

Desde mi punto de vista estos se objetivos se han cubierto bastante bien: se ha creado una aplicación capaz de gestionar usuarios y tareas de forma sencilla, mostrando la información sin redirecciones a urls ni tiempos de espera y de una forma limpia.

A nivel de eficiencia también se ha mejorado, ya que se han suprimido muchas validaciones en código que eran innecesarias, y se ha limpiado y reestructurado el código para seguir un único modelo de comunicación con el servidor para cualquier llamada.

Respecto a la seguridad, se han implementado filtros para restringir el acceso a recursos, se han eliminado usuarios y contraseñas de los parámetros y llevado a sesión, y se han implementado validaciones jquery en formularios para los usuarios. En este punto siempre se puede mejorar, y una posible mejora sería trabajar en esta línea.

Como conclusiones personales sobre este proyecto, podría agruparlas por fase:

- Análisis: es lo más importante del proyecto si nuestra idea es intentar reescribir código o realizar cambios, asegurarse que es exactamente lo que se está pidiendo.
- Diseño Web (estilos): ha sido una parte interesante, hoy en día existen multitud de frameworks y plugins que pueden ayudarte con ello, y lo que he aprendido es que siempre es mejor “buscar” a “inventar”, te das cuenta cuando pasas horas haciendo algo que resulta que ya está hecho y puedes utilizar importando una librería y poniendo una etiqueta.
- Diseño: esta parte ha sido tanto satisfactoria como frustrante. Lo que he aprendido es a no “reinventar la rueda”, no intentar desarrollar funciones complejas si ya están desarrolladas o probadas en otras librerías. También a organizar el mismo en pequeñas tareas y tener un flujo de trabajo a la hora de afrontar un problema.
- Documentación: nunca dejarla para el final. No hay nada peor que tener un producto y escribir de él cuando ya lo tienes desarrollado. Lo mismo con los comentarios dentro del código.

A nivel académico he adquirido sobre todo conocimientos en jquery y javascript, ya que no tenía ni la menor idea al comenzar el proyecto, por eso decidí basar el modelo de comunicación del sistema en llamadas jquery al servidor.

También he aumentado mis conocimientos en Java Servlets, seguridad, estilos web…, y sobre todo, he aprendido a ser autodidacta y buscar soluciones.

6.2 PLANES FUTUROS

Como posibles mejoras, podrían plantearse las siguientes líneas:

- Añadir tipos de tareas, esto sería sencillo ya que lo único que habría que implementar es la llamada y colocar el tipo de tarea en el desplegable.
- Parar tareas en ejecución.
- Aumentar el nivel de seguridad.
- Internacionalización de la aplicación.

BIBLIOGRAFÍA

http://www.w3schools.com/jquery/- JQuery tutorial

https://netbeans.org- Netbeans

http://es.wikipedia.org/wiki/Java_EE - Java EE - Wikipedia, la enciclopedia libre -

https://jquery.com/ - jQuery -

https://maven.apache.org/ - Maven - Welcome to Apache Maven -

http://logging.apache.org/log4j/2.x/ - Log4j - Log4j 2 Guide - Apache Log4j 2 -

http://getbootstrap.com/getting-started/- Bootstrap

http://www.tutorialspoint.com/jquery/jquery-filer.htm- JFiler

https://tomcat.apache.org/tomcat-5.5-doc/servletapi/javax/servlet/Filter.htm- Java Filter

http://www.tutorialspoint.com/servlets/servlets-file-uploading.htm- Servlet

http://www.w3schools.com/css/ - CSS

ANEXO I: MANUAL DE USUARIO

1 ZONA PRINCIPAL

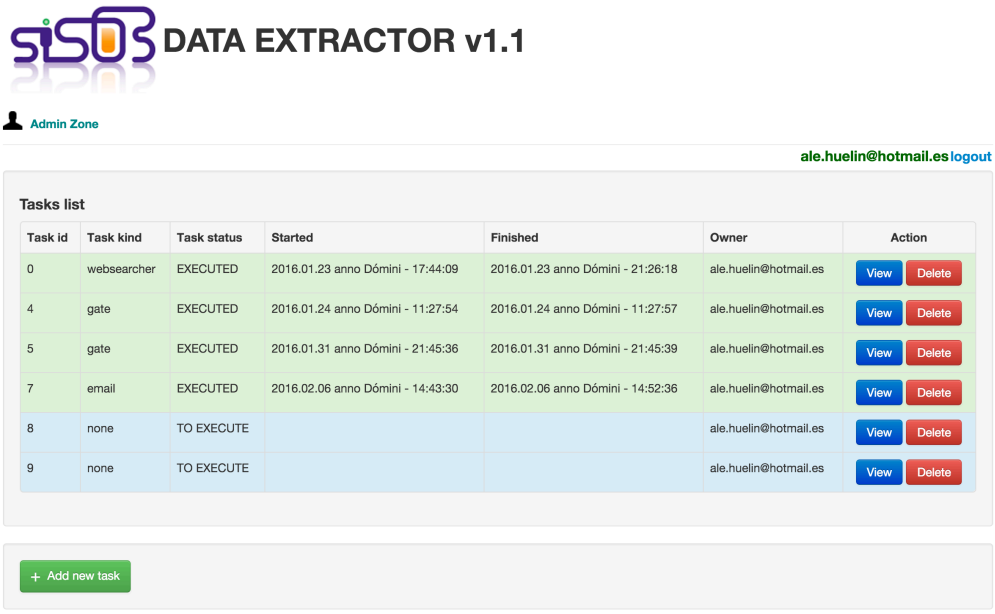


Ilustración 6: Main zone.

Es la pantalla principal de la aplicación, cuando hacemos login seremos redirigidos a aquí.

1.1.1 LISTADO DE TAREAS

Nada más entrar veremos nuestro listado de tareas, tenemos la posibilidad de acceder a ellas pulsando el botón “View”.

-Podremos eliminar las tareas, para ello podemos pulsar sobre el botón “Delete” y después de confirmar la operación, la tarea seleccionada quedará completamente eliminada del sistema.

-La otra opción posible desde esta pantalla es añadir una nueva tarea. Para ello sólo hay que pulsar sobre el botón “Add new Task”.

Desde esta pantalla principal podremos modificar los datos de nuestro usuario, para ello basta con pulsar sobre nuestra dirección de email, en la esquina superior derecha, y nos aparecerá la siguiente pantalla.

Existen 2 posibilidades, en función del perfil que tenga nuestro usuario tendremos unos privilegios u otros.

Usuario estándar:

Este usuario sólo podrá modificar su contraseña (password). La pantalla que le aparecerá será como la siguiente:

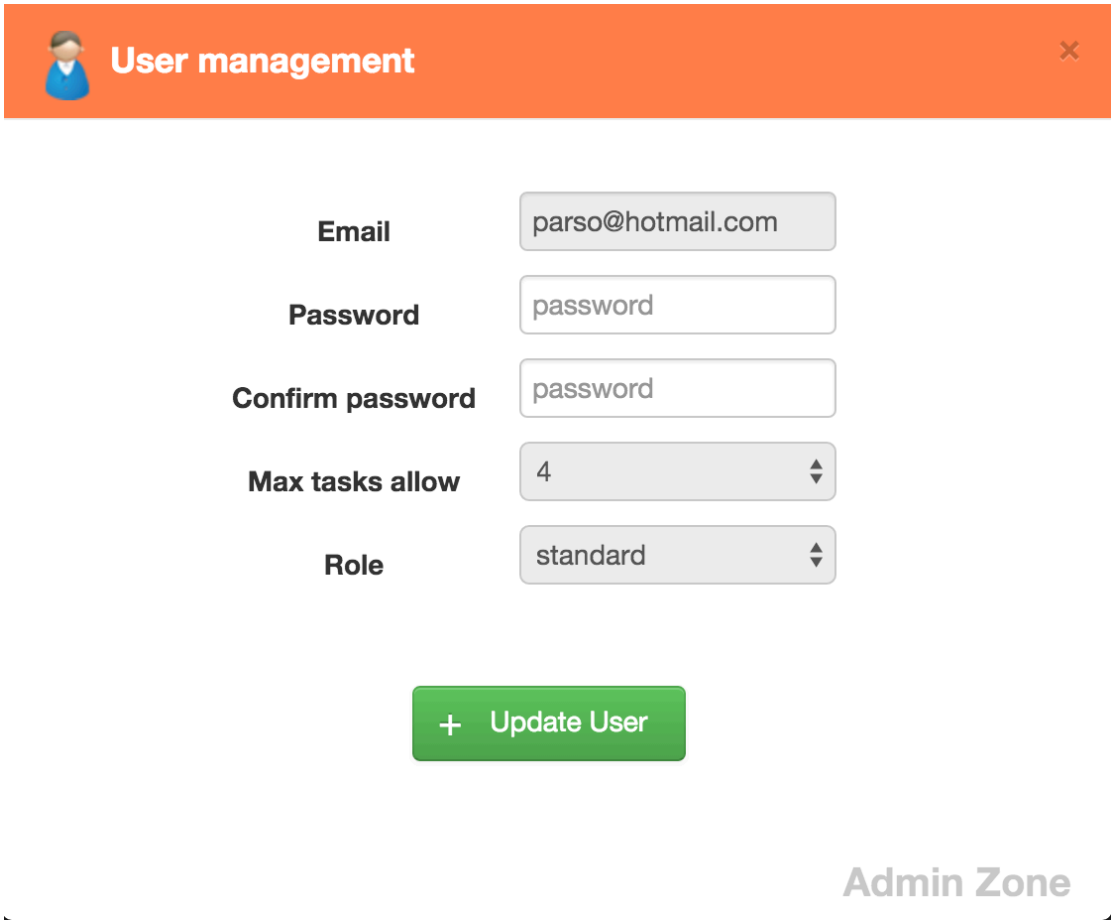


Ilustración 6.1: Edit window

Como podemos ver, sólo aparece editable el campo para la contraseña, los demás campos sólo son editables por un usuario administrador.

Usuario administrador:

Si hacemos clic en el mismo enlace pero habiéndonos autenticado con un perfil de administrador, la ventana sería la siguiente:

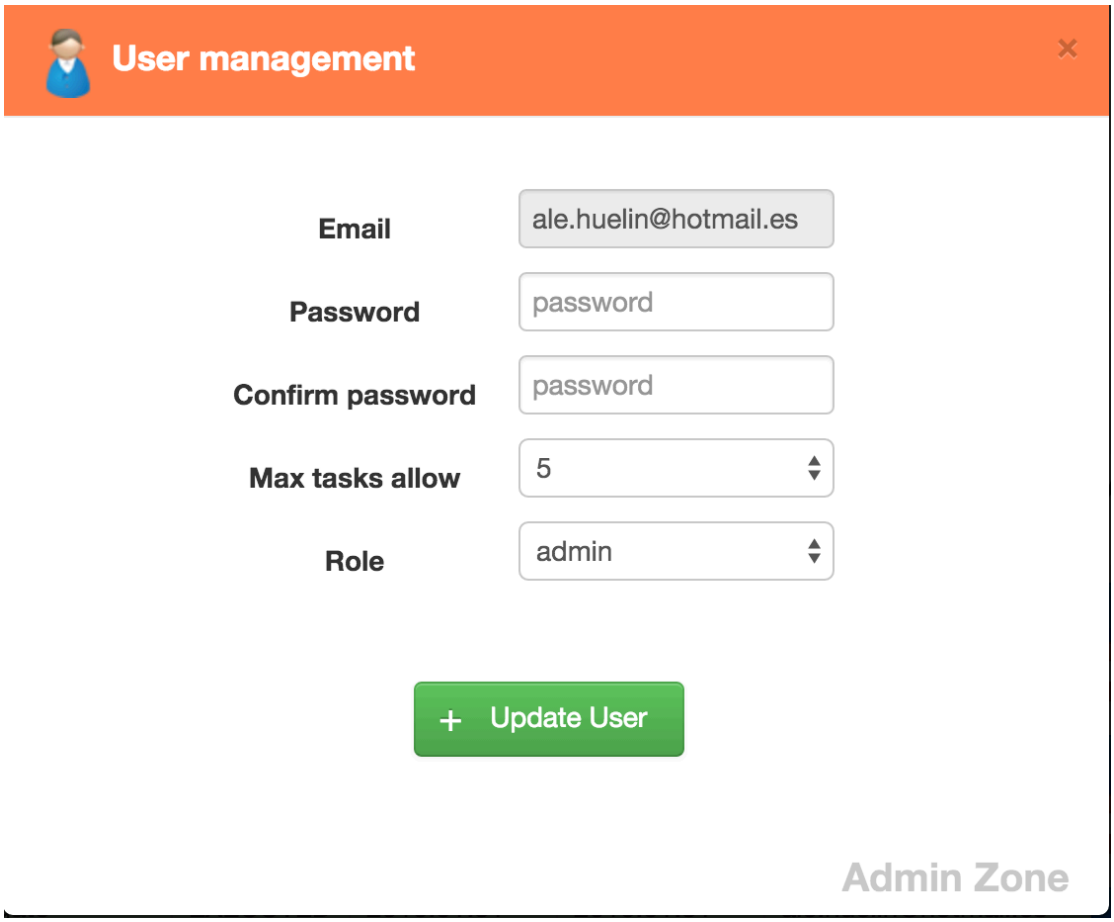


Ilustración 6.2: admin edition window.

Como podemos ver en la ilustración, ya aparecen los demás campos editables. Podemos establecer una nueva contraseña, el número máximo de tareas permitidas, que puede ser de 1 a 5 y el rol del usuario, estándar o administrador.

1.1.2 VER TAREA

Si hacemos clic en el botón “View” de cualquier tarea, nos aparecerá una ventana de carga, y posteriormente la tarea será cargada justo debajo del listado.

Después de pulsar “View”, veremos el siguiente mensaje:

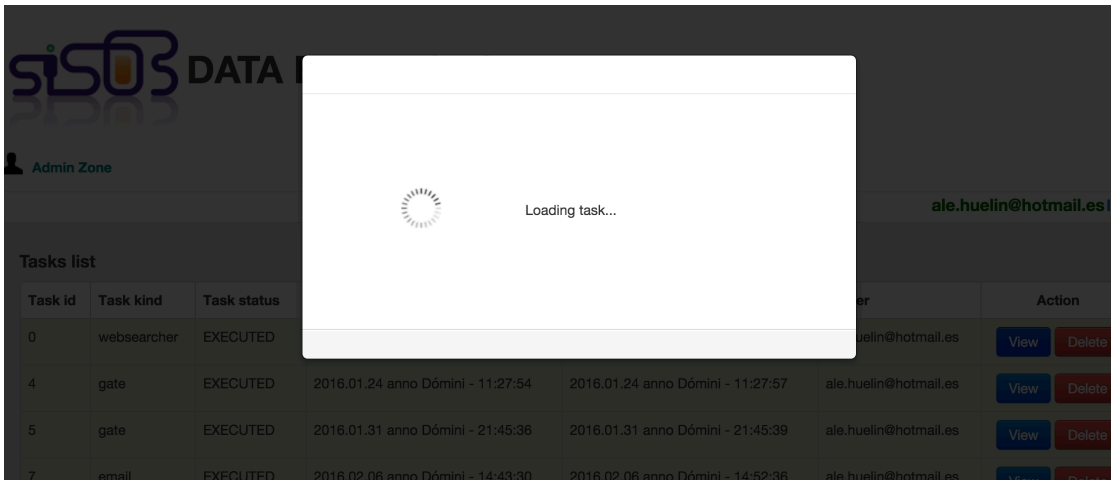


Ilustración 6.3: Loading modal.

Inmediatamente veremos que la tarea se ha cargado:

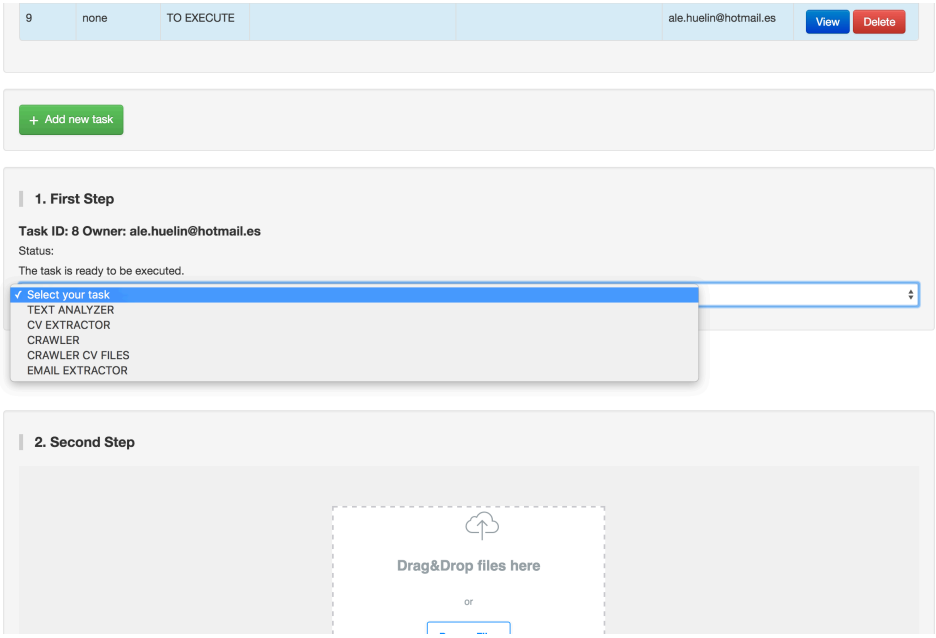


Ilustración 6.3: Loading modal.

El primer paso será elegir el tipo de tarea que queremos ejecutar. Hay 5 tipos de tareas:

1. **Text Analyzer:** El objetivo de esta tarea es extraer información personal de investigadores. Para ejecutar esta tarea se necesita como fichero de entrada un fichero .csv que contenga la siguiente información separada por punto y coma: identificador, nombre, apellidos y página web del investigador, como resultado devolverá otro Excel con una información que vendrá dada en tres bloques:
    - Datos personales: sexo, nacionalidad, email, teléfono.
    - Estudios universitarios: campo, calificaciones, institución, ciudad, país.
    - Profesión: cargo, fecha de comienzo y finalización, institución, país.
2. **CV Extractor:** Esta tarea tiene como objetivo extraer CVs de las páginas web de los investigadores. Para lanzas la tarea es necesario subir un fichero .csv que contenga la siguiente información separada por punto y coma: Nombre, primer apellido, segundo apellido, iniciales, especialidad, nombre de la universidad o institución del investigador, página web de la universidad o institución, página web del investigador, tipo de página web del investigador

(html) y cómo último parámetro, CV, para indicar que se quiere extraer el currículum.

3. **Crawler:** Esta tarea tiene como objetivo encontrar las páginas webs personales de los investigadores, el fichero de entrada que requiere esta tarea debe contener: nombre, apellidos y página web del investigador a buscar.

4. **Crawler CV Files:** El objetivo de esta tarea es encontrar cvs en formato pdf de un conjunto de investigadores dados, el fichero de entrada que utilizamos en este caso contiene: nombre del investigador, apellidos, inicial del nombre, especialidad, nombre de la institución, url de la institución en la que trabaja.
5. **Email extractor:** El objetivo de esta tarea es la extracción de los emails de las páginas de los investigadores o de sus CVs. Requiere un fichero de entrada que debe contener la siguiente información: Nombre, apellidos, iniciales, especialidad, nombre de la universidad o institución, página web de la universidad o institución, tipo de la página (puede ser html o pdf, según se quiera buscar en la web o en un cv en formato pdf).

Elegimos una de las 5 y continuamos con el siguiente paso.

Después de elegir tarea, el sistema mostrará un cuadro de texto que contendrá una explicación según el tipo de tarea sobre cómo ejecutarla, por ejemplo si elegimos la de “email extractor”:

Nos aparecerá lo siguiente:

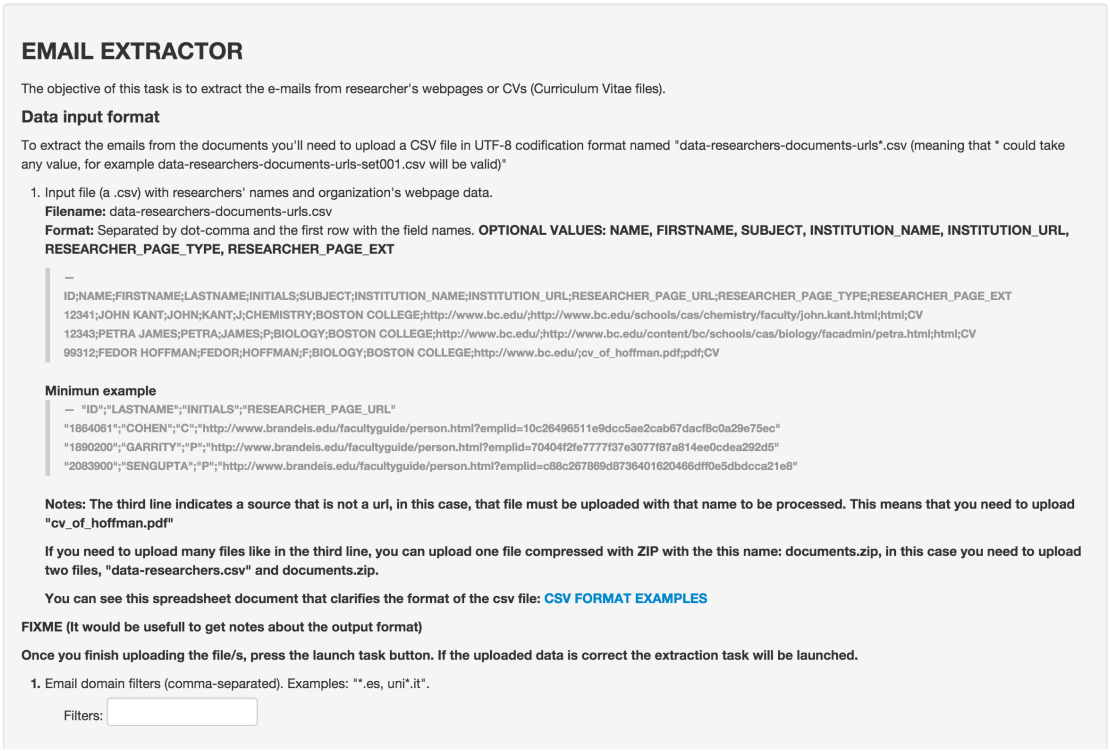


Ilustración 6.3: Instructions.

Lo que vemos en la ilustración es una breve descripción de la tarea a ejecutar, y los pasos a seguir para lanzarla, en este caso la tarea es extracción de emails de páginas de investigadores o currículums:

Para ejecutarla, necesitamos un fichero de entrada, este fichero es un archivo .csv, que debe llamarse “data-researchers-documents-url*.csv”. Este fichero debe tener codificación UTF-8.

Básicamente lo que hay que hacer es subir un Excel, según el tipo de tarea se necesita un formato para los Excel que debemos subir. Por ejemplo, para continuar con este ejemplo de tarea del tipo “email extractor”, se necesita subir un fichero con nombre data-researchers-documents-url*.csv.

Procedemos con el segundo paso y subimos:

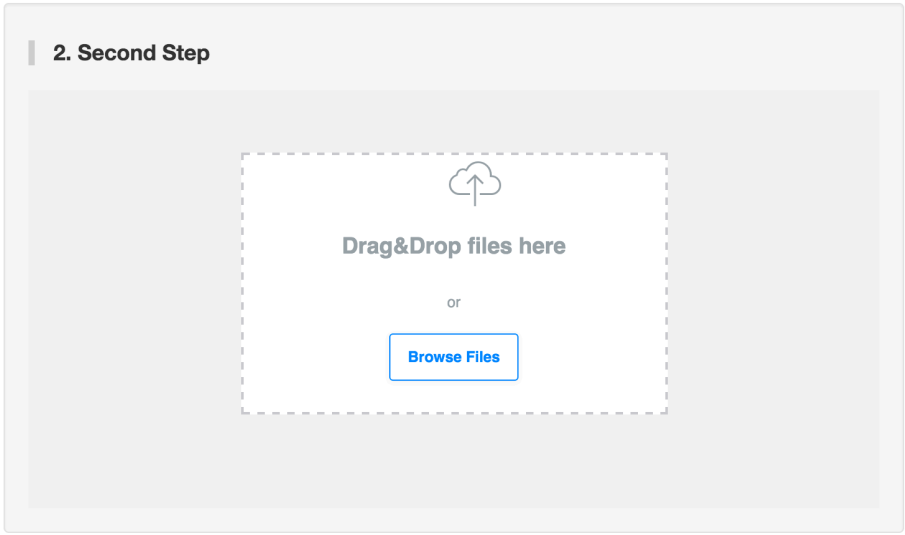


Ilustración 6.4: Instructions

Podemos tanto subirlo mediante el explorador de archivos como arrastrando sobre la cuadrícula:

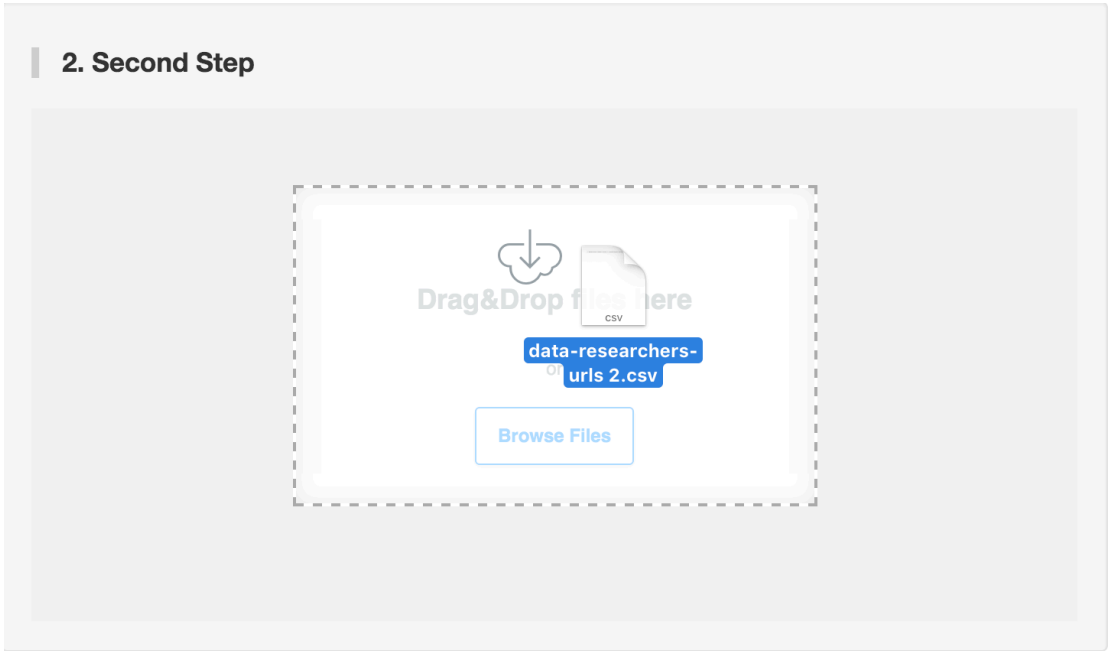


Ilustración 6.5: Instructions

Si se ha subido correctamente debe quedarnos como en la siguiente imagen:

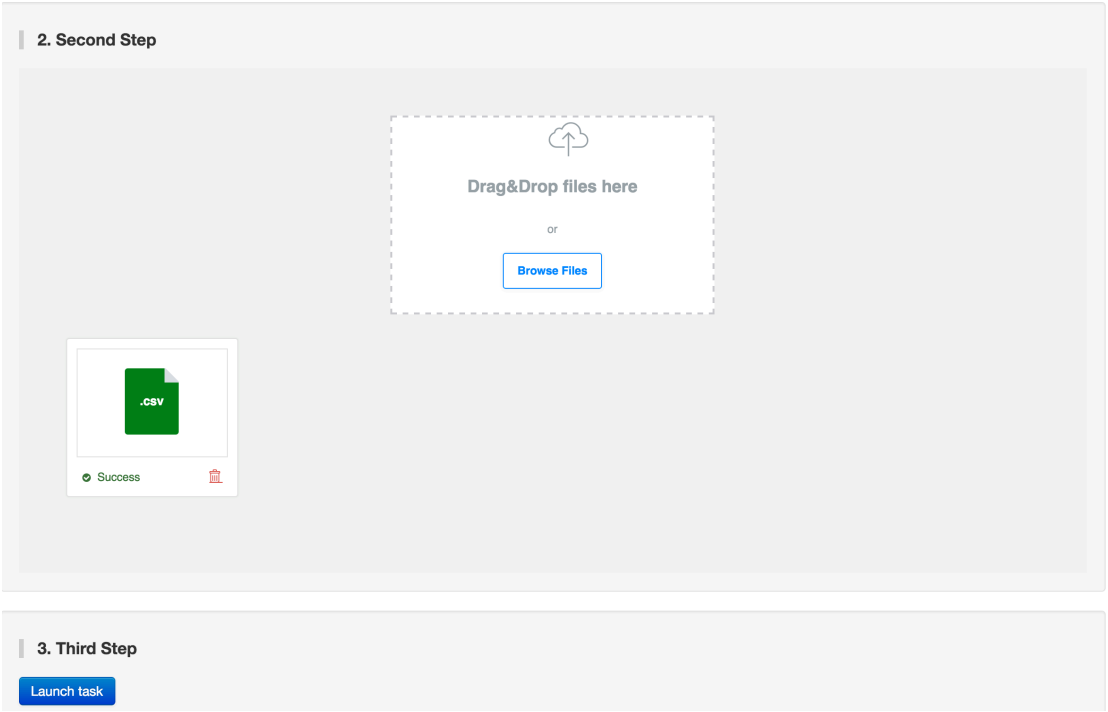


Ilustración 6.6: Instructions

Una vez hecho esto podemos proceder a lanzar la tarea pulsando “Launch task”, seguidamente recibiremos un mensaje de confirmación para saber si todo fue bien, y ya sólo queda esperar. El listado de tareas actualizará el estado en el que se encuentra la tarea automáticamente.

En amarillo aparece la tarea que está en ejecución:

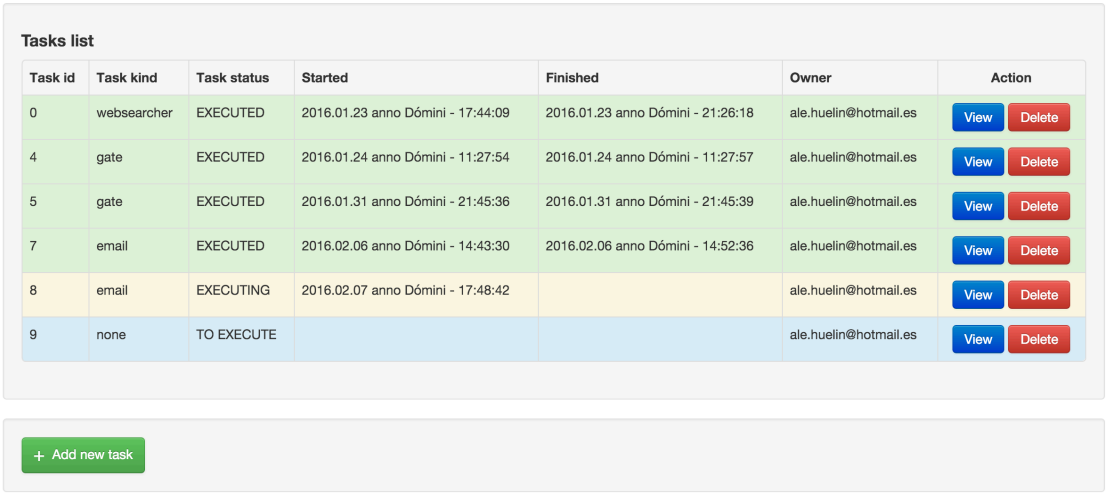


Ilustración 6.7: Instructions

Una vez ejecutada la tarea, podemos pulsar “View” y podremos acceder a los resultados de la misma, estos se cargarán debajo del listado, igual que para ejecutarla. Una vez veamos los ficheros generados como resultado de la ejecución de la tarea, podremos descargarlos pulsando sobre los mismos.

1.1.3 ADMIN ZONE

Esta zona sólo será visible para los usuarios con perfil de administrador. La encontraremos en la esquina superior izquierda de la pantalla principal.

Si pinchamos sobre ella, podremos acceder y nos encontraremos la siguiente pantalla:

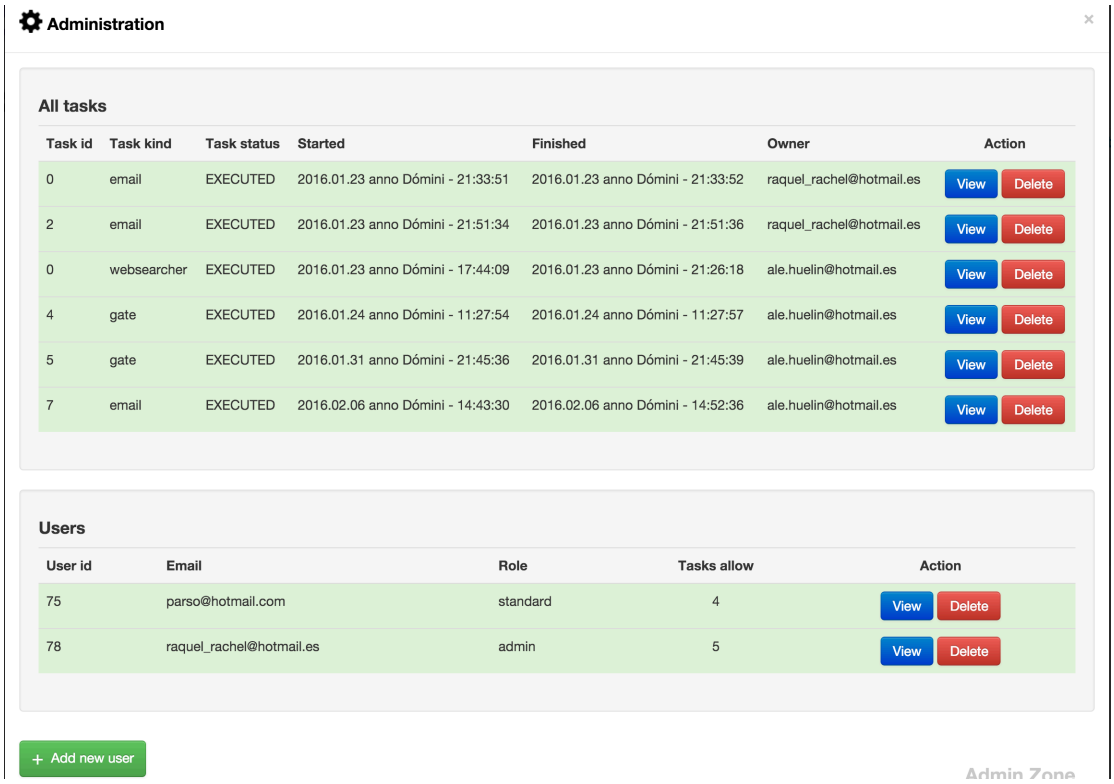


Ilustración 6.8: Admin Zone.

Desde esta pantalla podemos gestionar tareas y usuarios:

Podrá ver cualquier tarea y posteriormente lanzarla si queremos, eliminar cualquier tarea y respecto a usuarios podrá crear, editar y eliminarlos.

Crear usuario

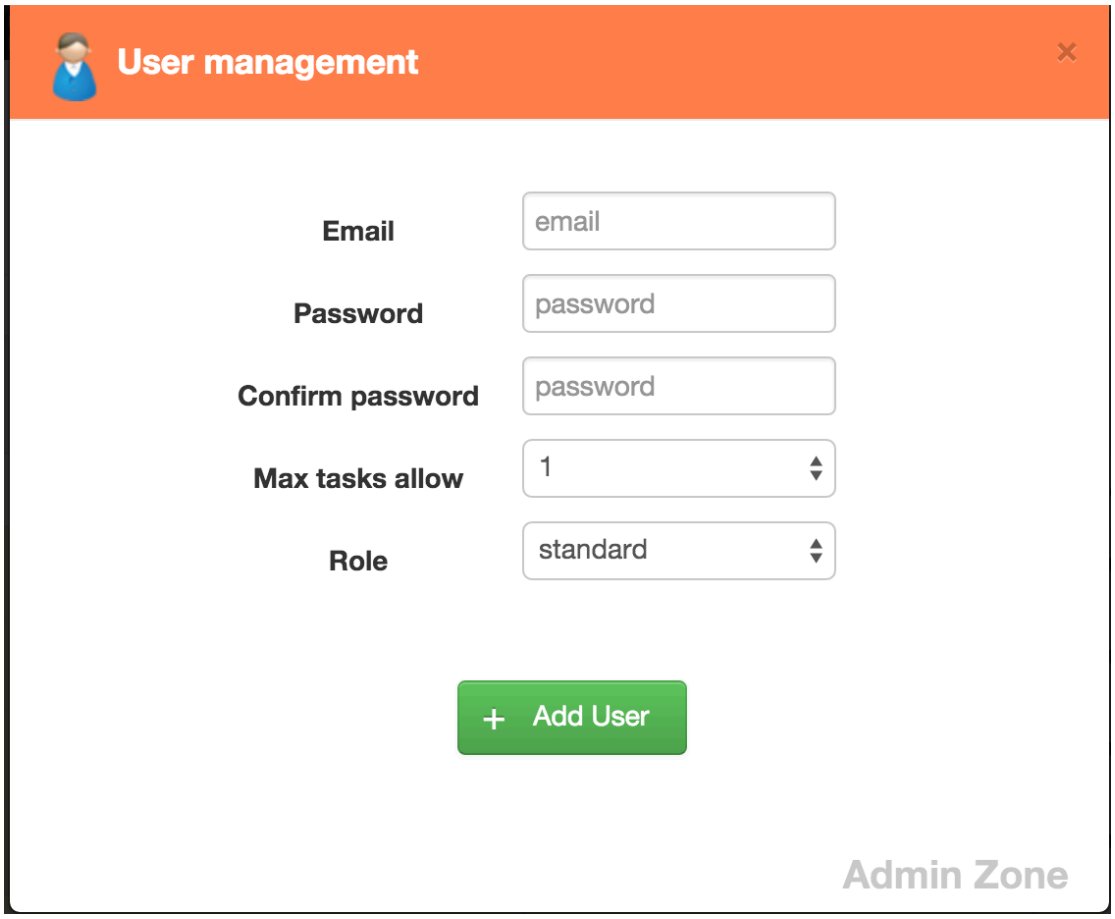


Ilustración 6.9: New user.

Para crear un nuevo usuario es necesario completar todos los campos correctamente y pulsar el botón verde. A continuación recibiremos un mensaje de información y el usuario aparecerá en el listado.

Para editar un usuario basta con pulsar “View”, y se nos abrirá una pantalla con los datos del usuario que podremos modificar.